# Studies on Algorithms and Analyses for Pickup and Delivery Problems( Dissertation_全文 )

AUTHOR(S):

Nakao, Yoshitaka

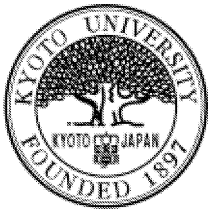
# Studies on Algorithms and Analyses for Pickup and Delivery Problems

Yoshitaka NAKAO

# STUDIES ON ALGORITHMS AND ANALYSES FOR PICKUP AND DELIVERY PROBLEMS

Yoshitaka NAKAO

Department of Applied Mathematics and Physics
Graduate School of Informatics
Kyoto University
Kyoto 606-8501, Japan




March, 2010

Doctoral Dissertation

submitted to Graduate School of Informatics, Kyoto University

in partial fulfillment of the requirement for the degree of

DOCTOR OF INFORMATICS

(Applied Mathematics and Physics)

# Preface

Improving vehicle routes is a fundamental subject in decreasing physical distribution cost and reducing CO2 emission for most manufacturing and distribution companies. The percentage of physical distribution cost to sales is 4.9% in Japan and 9.3% in the United States, which is higher than cost for research and developments. In the environmental point of view, most of advanced nations are required to reduce the amount of CO2 emission more than 5% comparing to that in 1990 due to Kyoto protocol.

The vehicle routing problem (VRP) and the pickup and delivery problems (PDP) are known as one of the models that aim to improve vehicle routes. VRP is defined in the following way. Given a set of vehicles starting and ending at a depot and a set of customers with their demand, the problem asks to find a route for each vehicle such that the total travel cost is minimized under the restrictions that all customer demands are met and each customer is visited exactly once. The latter restriction indicates that each customer demand cannot be split. PDP is an extension of VRP that handles pickup and delivery of loads between pairs of customers. Each transportation request must be picked up at a predetermined customer and delivered to an another predetermined customer. In PDP, the actions of pickup and delivery must be done by the same vehicle, and the action of pickup must be done before that of delivery. The total quantity of loads cannot exceed the vehicle capacity any time. Note that VRP is a special case where all pickup customers are located on the same place (i.e., depot). There exist many variants of VRP and PDP, such as VRP with time windows (VRPTW), the split delivery VRP (SDVRP), PDP with time window (PDPTW), and PDP with transfer (PDPT). They are all known to be NP-hard. It is strongly believed that an NP-hard problem cannot be solved in polynomial time of the input size. Thus solving an NP-hard problem exactly may necessitate enumerating an essential portion of the set of all solutions, whose number increases exponentially as problem size grows. However in most practical applications, we do not need exact optimal solutions and are satisfied with approximately good solutions.

Most of manufacturing and distribution companies apply the following two approaches in order to decrease distribution cost. One is finding the cheapest routing schedule that obeys given routing constraints using the existing facilities. Several algorithms for VRP, PDP and

their variations have been studied and there have been hundreds of successful applications in many industries. One contribution in this thesis is that proposing a fast algorithm for a variation of VRP, a discrete-type split delivery vehicle routing problem, in which delivery goods for a customer consist of a set of items, each item is required to be serviced by exactly one vehicle, and each customer is allowed to be visited more than once.

The other approach for decreasing cost is making a small change to the current routing rules and the existing facilities in the entire distribution system. The maximum saving by this approach is usually larger than that by finding the cheapest routes with the current routing rules and facilities. Thus it is usually required to know the possible effect by such a change through an estimation of the performance of the resulting system before we actually introduce it to the current system. In this thesis, we analyze the maximum saving that can be brought by relaxing some constraints, and especially examine the maximum ratio of the optimal value of PDP to that of PDPT, and PDPCMT to that of PDP. For both analyses, we present not only upper bounds but also instances that achieve lower bounds.

Due to the enhancement of information technology, efficient algorithms for VRP and PDP are applied to distribution systems in a real world, and they have been making drastic improvement on decreasing distribution cost and CO2 emission. Furthermore excellent theoretical analyses would contribute to making an important decision on changes to the current routing rules and the existing facilities. The author hopes that the work contained in this thesis will be helpful to the study and the application to a real world in this field.

Kyoto, March 2010

Yoshitaka Nakao

# Acknowledgement

First of all, I am heartily grateful to my adviser, Professor Hiroshi Nagamochi for his enthusiastic guidance, discussion and persistent encouragement. He gave me opportunity to enter doctor's program at graduate school of Kyoto University. Even in his very busy moment, he takes time to supervise my study, in particular, to supervise this dissertation. He gave me earnest guidance to get ideas for algorithms and analyses. Furthermore he commented very carefully on the manuscript, which significantly improved the accuracy and quality of the expression. Without his considerable help, none of my entrance to doctor's program and this work could have been completed.

A great deal of gratitude goes to Professor Toshihide Ibaraki who gave me invaluable advice and encouragement during my studies in master's program. His excellent work attracted my attention to the field of combinatorial optimization.

I wish to express my gratitude to Professor Liang Zhao who gave me so much supports to study in Nagamochi Lab. The gratitude is extended to Professor Mutsunori Yagiura and Professor Hideki Hashimoto, who gave me necessary information and concerns for vehicle routing problems. I also thank all members in the set of Nagamochi Lab.

I am also thankful to Professor Masao Fukushima and Professor Yoshito Ohta for serving on my dissertation committee.

I am deeply grateful to members in Mathematical Science Department in Canon IT Solutions Inc., who have given me helpful and enthusiastic guidance in the field of business that incorporates combinatorial optimization since I joined the company. The experience during my work at the company greatly helped to complete this dissertation.

Finally, I would like to express my gratitude to my wife and daughter. I also want to thank all my relatives, especially parents, grand mothers, sister, and parents-in-law. Thanks for supporting and caring me so long.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Background

In these decades, it is a fundamental subject for most manufacturing and distribution companies to improve physical distribution and logistics for the purpose of decreasing physical distribution cost and reducing CO2 emission.

We first focus on distribution cost. The percentage of physical distribution cost to sales is 4.9% in Japan and 9.3% in the United States [44]. In Japan, the percentage of physical distribution cost to sales in foods industry is beyond 10%, and the percentages of distribution cost to sales in industries of glass, cement, paper and pulp are also high, which are approximately 10%. Comparing to the percentage of research and development cost to sales, which is about 3.9% in manufacturing companies, the percentage of physical distribution cost is considerably high. In the breakdown of the distribution cost, the shipping cost is 59.1%, the warehousing cost is 16.2%, and cost for others is 24.7%. Thus the shipping cost surpasses other costs. When comparing quantity of physical distribution in the segments of transportation such as automobiles, airplanes and ship, the percentage of physical distribution cost by automobiles is the highest, which is 90% by ton, and 54% by ton times kilometer. The quantity of transportation by automobile has been increasing significantly. For example, the quantity by ton times kilometer has increased by 10% in recent ten years, and by 75% in twenty years. This is due to the rapid and continuous development of expressways and equipment of automobile. Thus it is considerably required to reduce delivery cost that is caused by automobile.

We next focus on environmental problems. Annual report on the environment in Japan [55] reports that the percentage of quantity of CO2 emission by automobile transportation to total quantity is about 20%. Furthermore not only CO2 emission, but also emission of oxide of nitrogen (NOx) by automobile has anxiety problems, for example global warming and health hazard such as carcinogenesis. Thus improving routes by automobile contributes not only to decreasing distribution cost but to improving environment on the earth.

The vehicle routing problem (abbreviated as VRP) and the pickup and delivery problem (PDP) are known as typical models that aim to decrease delivery cost. VRP is defined in the following way. Given a set of vehicles starting and ending at a depot and a set of customers with their demand, the problem asks to find a route for each vehicle such that the total travel cost is minimized under the restrictions that all customer demands are met and each customer is visited exactly once. The latter restriction indicates that each customer demand cannot be split. Each vehicle has a capacity and the total quantity of loads cannot exceed the vehicle capacity. Since a paper by Dantzig and Ramser [22] appeared in 1959, VRP has been intensively and continuously studied [18, 32, 37, 48, 88, 89]. There exist many variants of VRP, such as the *vehicle routing problem with time windows* (VRPTW) [25, 26, 42, 82, 83, 86], the *multiple depot vehicle routing problem* (VRPMD) [23], the *vehicle routing problem with multiple trips* (VRPMT) [70], the *vehicle routing problem with backhauls* (VRPB) [90], the vehicle routing problem with pickup and delivery (VRPPD) [24, 59], and the split delivery vehicle routing problem (SDVRP) [3, 4, 5, 8, 17, 28, 29, 30, 34, 35]. VRP and its variants are known to be NP-hard.

The pickup and delivery problem (abbreviated as PDP) is an extension of VRP that handles pickup and delivery of loads between pairs of customers [76]. Each transportation request must be picked up at a predetermined customer and delivered to an another predetermined customer. PDP introduces two side constraints. One is a *coupling constraint* that the actions of pickup and delivery must be done by the same vehicle. The other is a *precedence constraint* that the action of pickup must be done before that of delivery. The total quantity of loads cannot exceed the vehicle capacity any time. Note that VRP is a special case where all pickup customers and depots are located on the same place. PDP and its variants also are known to be NP-hard as it contains the vehicle routing problem. Many heuristics and metaheuristic algorithms have been developed for the problem [51, 63, 68]. Several variants of PDP are known such as the pickup and delivery problem with time windows (PDPTW) and the pickup and delivery problem with transfer (PDPT). PDPT is a problem such that each request can be served by more than one vehicle by dropping a load at a *transshipment point* and picking it up by another vehicle, that is, the coupling constraint for PDP is relaxed [19, 56, 78].

Most distribution and manufacturing companies usually take one of the following two approaches in order to decrease physical delivery cost. One approach is that finding a cheaper routing schedule than the current one by modeling the current routing constraints and the existing facilities to a problem of VRP, PDP or their variations and solving the problem by using algorithms which are designed sometimes in a sophisticated way. In this case, companies obey given routing constraints and do not change any facilities. In the past four decades, many exact and heuristic algorithms have been proposed, which will be overviewed in Section 1.4.

The other approach is that changing the current routing rules and the existing facilities in the entire distribution systems. For example, constructing a transshipment warehouse where trucks can transfer a part of loads on the vehicles each other may reduce total travel cost. In most cases, the maximum saving that can be achieved by making a change to the routing rules and facilities is usually rather large comparing to the best choice of routing schedules with current routing rules and facilities. However, it is usually required to know the possible effect by such a change through an estimation of the performance of the resulting system before we actually introduce it to the current system. It would be considerably desirable if we have some common knowledge on the maximum possible cost saving by each type of constraints on routing and functions of facilities before we select promising changes from many candidates of changes without conducting accurate and expensive estimations of their performances. In Section 1.5, we review theoretical analyses that have been achieved in the past.

## 1.2 Basic Definition

This section gives basic definitions that are used to introduce variants of PDP in the next section. Given a real number $x$, let $\lfloor x \rfloor$ denote the maximum integer that is not beyond $x$, while $\lceil x \rceil$ denotes the minimum integer that is not below $x$.

Depots and customers to be visited by vehicles are represented by vertices in an edge-weighted complete digraph with a vertex set $V$, which is given by distance functions $d(u, v)$ for all ordered pairs of vertices $u$ and $v$. Distance $d(u, v)$ is a nonnegative real number, and in general asymmetric, i.e., $d(u, v) \neq d(v, u)$ may hold. Distance $\{d(u, v) \mid u, v \in V\}$ may not satisfy the Triangle Inequality.

An instance consists of a set $Q$ of *depots*, a set $R$ of *requests*, and a vehicle capacity $c > 0$. Each request $r = \{r^+, r^-\} \in R$ consists of a *pickup action* $r^+$, a *delivery action* $r^-$, and a quantity $q(r)$ of loads for the request. That is, quantity $q(r)$ of loads is required to be picked up at a specified vertex where pickup action $r^+$ is taken and to be delivered to a specified vertex where delivery action $r^-$ is taken. We may denote the vertex where request $r$ is picked up by $r^+$ (resp., delivered by $r^-$), and call the vertex $r^+$ a *pickup point* (resp., $r^-$ a *delivery point*). Let $A = \{r^+, r^- \mid r \in R\}$, i.e., the set of all actions for $R$. Let $p$ be the number of requests in $R$.

Each vehicle must start from a depot in $Q$, and return to the same depot after serving some of the requests in $R$. A *route* is the requests assigned to a vehicle, and is represented by the sequence $\sigma = [a_0, a_1, a_2, \ldots, a_m, a_{m+1}]$ of actions of the requests in the order that the vehicle serves, where $a_1, a_2, \ldots, a_m \in A$ ($m$ is an even integer) and $a_0, a_{m+1} \in Q$. The travel

cost of route $\sigma$ is defined to be

$$cost(\sigma) = \sum_{0 \le i \le m} d(a_i, a_{i+1}).$$

A *solution* $s$ is a set of routes that serve all requests in $R$, and its cost $cost(s)$ is defined to be the sum of the travel costs of the routes $\sigma$ in solution $s$. The objective is to find a minimum cost solution, and we assume that any number of vehicles is available.

Given an instance $I$ and a problem PRB, we call a solution to PRB with instance $I$ a *PRB solution* to $I$, and call a route in PRB solution with instance $I$ a *PRB route* to $I$.

## 1.3 Variants of PDP and VRP

This section introduces variants of PDP. Constraints on serving requests depend on types of problems. Fig. 1.1 illustrates types of routes used for the routing problems which we explain in this section. Dashed lines correspond to subsequences with no loads while bold lines correspond to those with loads.

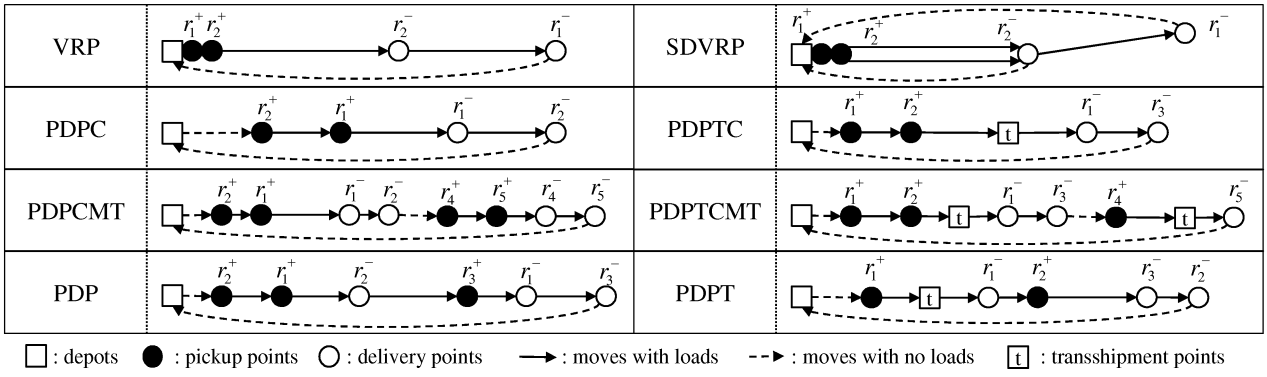



Figure 1.1: Illustration of types of routes for routing problems.

We start with defining PDP.

**Pickup and Delivery Problem (PDP):**
**Input:** An instance $I = (Q, R, c)$.
**Output:** A minimum cost solution that satisfies the following constraints (see Fig. 1.1):

(a) the total quantity of loads during a route does not exceed vehicle capacity $c$ at any time;

(b) each request $r$ is served by exactly one vehicle, and the actions $r^+$ and $r^-$ are taken exactly once by the vehicle;

(c) (*coupling constraint*) actions $r^+$ and $r^-$ of each request $r$ must appear in the same route, and no request $r$ is allowed to be temporarily dropped at any vertices; and

(d) (*precedence constraint*) for each request $r$, action $r^+$ must be taken before action $r^-$.

We next introduce an important variant of PDP; *multi-trip PDP with consecutive pickups and deliveries* (PDPCMT) which we encounter in many practical cases such as distribution of steel products, foods, drinks, etc. PDPCMT is described as follows.

**Multi-Trip PDP with Consecutive Pickups and Deliveries (PDPCMT):**
**Input:** An instance $I = (Q, R, c)$.
**Output:** A minimum cost solution that satisfies constraints (a), (b), (c), (d), and the following constraint:

(e) once a delivery action begins, pickup actions cannot be taken until all of the loads on the vehicle are delivered (see Fig. 1.1).

A route satisfying constraint (e) is a sequence of subsequences such that first a vehicle with no loads on it takes pickup actions for some requests and takes delivery actions for these requests. We call such a subsequence a *trip*.

PDPCMT is called *PDP with consecutive pickups and deliveries* (PDPC) if each route is required to consist of a single trip. Thus PDPC is presented as follows.

**PDP with Consecutive Pickups and Deliveries (PDPC):**
**Input:** An instance $I = (Q, R, c)$.
**Output:** A minimum cost solution that satisfies constraints (a), (b), (c), (d) and the following constraint:

(f) once a vehicle take a delivery action, it is not allowed to take any pickup action during the same route.

To describe the vehicle routing problem (VRP) [48, 89], we introduce the following condition:

(g) the travel cost between any two pickup vertices is 0.

VRP is described as follows.

**Vehicle Routing Problem (VRP) :**
**Input:** An instance $I = (Q, R, c)$ with condition (g).
**Output:** A minimum cost solution that satisfies (a), (b), (c), (d) and (f).

In VRP, all pickup points and depots for all vehicles correspond to a unique vertex. Thus

vertices in VRP consists of one depot and customers. Note that customers in VRP correspond to delivery points in PDP.

In the split delivery vehicle routing problem (SDVRP), the restriction that each customer is visited exactly once is relaxed [3, 4, 5, 8, 17, 28, 29, 30, 34, 35]. Thus SDVRP drops constraint (b) from constraints for VRP. Demand for each customer may beyond vehicle capacity. SDVRP is described as follows.

**Split Delivery Vehicle Routing Problem (SDVRP):**
**Input:** An instance $I = (Q, R, c)$ with condition (g).
**Output:** A minimum cost solution that satisfies constraints (a), (c), (d) and (f).

Fig. 1.2 illustrates an example of a VRP solution and an SDVRP solution that is presented in [29]. In Fig. 1.2, vehicle capacity $c$ is given by $c = 5$. The left side solution in Fig. 1.2 is a VRP solution where each customer demand is served by one vehicle, and the total travel cost is $6d$. The right side solution is an SDVRP solution where the three customer demands are served by two vehicle, and the total travel cost is $4d + 2\epsilon$. The example shows that when $\epsilon$ goes to 0, the minimum cost of a VRP solution is almost 1.5 times more expensive than the minimum cost of an SDVRP solution.

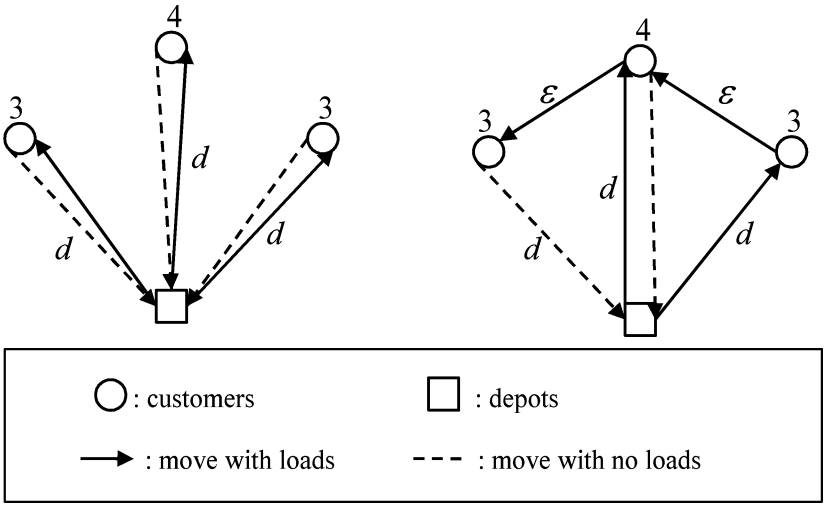



Figure 1.2: A example of VRP and SDVRP solutions.

We next introduce a new variant of SDVRP. The original SDVRP admits splitting a demand into pieces of any real size. However, in this variant, loads for a request consists of a set of items, each of which cannot be split, where items may have different sizes. We encounter this type of problem more frequently than SDVRP since most loads consist of portions of items such as carbon boxes. We call this discrete-type of SDVRP the *discrete*

*split delivery vehicle routing problem* (DSDVRP).

**Discrete Split Delivery Vehicle Routing Problem (DSDVRP):**
**Input:** An instance $I = (Q, R, c)$ with condition (g).
**Output:** A minimum cost solution that satisfies constraints (a), (c), (d) and the following constraint:

(h) loads for a request consists of a set of items, each of which cannot be split.

The pickup and delivery problem with transfer (PDPT) introduces a set of transshipment points to PDP. This thesis assumes that the number of transshipment points is one in order to make it possible to analyze the ratio of travel cost of PDPT to that of PDP in Chapter 4. Let $t$ denote the transshipment point. At transshipment point $t$, each vehicle is allowed to temporarily drop some of the loads on it and the loads will be picked up later by the vehicle or some other vehicle [56, 62, 78]. This indicates that PDPT is obtained from PDP by dropping constraint (c) (see Fig. 1.1).

**PDP with Transfer (PDPT):**
**Input:** An instance $I = (Q, R, c, t)$.
**Output:** A minimum cost solution that satisfies constraints (a), (b) and (d).

The use of a transshipment point can yield solutions with smaller travel cost. Fig. 1.3 shows an instance, where the travel cost of PDPT route is smaller than that of PDP solution. In Fig. 1.3, there exist two depots $p_1$ and $p_2$, two pickup points $r_1^+$ and $r_2^+$, two delivery points $r_1^-$ and $r_2^-$, and transshipment point $t$. The travel cost satisfies $d(p_1, r_1^+) = d(p_2, r_2^+) = 0$, $d(r_1^+, r_2^+) = d(r_1^+, r_1^-) = d(r_2^+, r_2^-) = d(r_2^+, r_1^-) = 2$, and $d(t, v) = 1$ for all $v \in \{p_1, p_2, r_1^+, r_2^+, r_1^-, r_2^-\}$. There are four requests $\{r_1^+, r_1^-\}, \{r_1^+, r_2^-\}, \{r_2^+, r_1^-\}$, and $\{r_1^+, r_2^-\}$ with quantity $c/2$, and two vehicles whose depot are $p_1$ and $p_2$ are available. In a PDP solution, a vehicle whose depot is $p_i$, $i = 1, 2$, picks up two requests $\{r_i^+, r_1^-\}$ and $\{r_i^+, r_2^-\}$, and delivers them to their delivery points. In a PDPT solution, a vehicle whose depot is $p_i$ picks up two requests $\{r_i^+, r_1^-\}, \{r_i^+, r_2^-\}$, transfers one of the requests at $t$ so that it delivers requests $\{r_1^+, r_i^-\}, \{r_2^+, r_i^-\}$, and it finally delivers them to delivery point $r_i^-$. The travel cost to PDPT solution is 8, which is smaller than that of PDP solution with travel cost 12.
We finally introduce PDPTC and PDPTCMT that allow transfers to PDPC and PDPCMT, respectively.

**Multi-Trip PDP with Transfer with Consecutive Pickups and Deliveries (PDPTCMT):**
**Input:** An instance $I = (Q, R, c, t)$.
**Output:** A minimum cost solution that satisfies constraints (a), (b), (d) and (e).

**PDP with Transfer with Consecutive Pickups and Deliveries (PDPTC):**

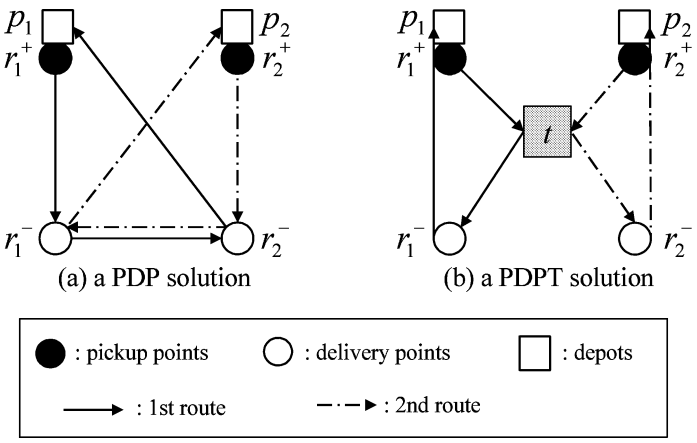



Figure 1.3: A example of PDP and PDPT solutions.

**Input:** An instance $I = (Q, R, c, t)$.
**Output:** A minimum cost solution that satisfies constraints (a), (b), (d) and (f).

In PDPTCMT and PDPTC, a vehicle is allowed to visit transshipment point $t$ after the vehicle visits all pickup points of its requests and before it visits the corresponding delivery points during its trip.

## 1.4 Algorithms for VRP and PDP

As described in Section 1.1, one approach to reduction of physical distribution cost is finding a cheaper routing schedule that obeys the current routing constraints by using algorithms. In this section, we overview algorithms for VRP, SDVRP, and variants of PDP that have been proposed before.

### 1.4.1 Algorithms for VRP

We first overview algorithms for VRP. Heuristics proposed for VRP can be classified into two main classes:

- classical heuristics (construction algorithms) that are proposed mostly between 1960 and 1990;
- metaheuristics whose growth has occurred in the last two decades.

Most standard construction and improvement procedures in use today belong to the first class. These methods produce good quality solutions within short computation time. For example, the saving algorithm [18] by Clarke and Wright is one of the well known heuristic

for VRP. This algorithm starts with a solution where each customer is visited by one vehicle. Then it iterates selecting two routes such that savings attained by merging routes is the maximum, and merging them. The insertion algorithm [82] starts from an empty set of routes and unrouted customers. It iterates inserting an unrouted customer into one of the current routes so that the cost increased by the insertion is the minimum until all of the customers are inserted into any of routes. Gillet and Miller [39] proposed a sweep algorithm for the Euclidian VRP with a unit capacity. Positions of customers are given in polar coordinates, and the depot is located in the center. Customers are ordered in an ascending order of their arguments. The sweep algorithm divides a set of customers in such a way that the total quantity of requests for each subset which is served by one vehicle does not exceed the capacity. The detail of those three algorithms will be described in Chapter 2.

Cluster-first/route-second algorithm uses the following method:

- firstly assign each customer to a vehicle (clustering),
- secondly solve the traveling salesman problem (TSP) for customers that are assigned to each vehicle.

Fisher and Jaikumar [32] split the clustering problem into two parts: firstly assigning a "seed" customer to each vehicle and secondly assigning all customers which have not yet assigned to the cheapest vehicle according to some objective function.
The general idea of route-first/cluster-second algorithm is as follows:

- construct a TSP tour for all customers (not including the depot),
- divide the tour into subtours such that all subtours can be assigned to a vehicle.

Bowerman, Calamai and Brent Hall [11] use space filling curves to find a TSP tour. Their approach assumes that customers are located on the Euclidean space. They divide a unit square where customers are located recursively into four smaller squares, and find a curve which visits all areas in the unit square. According to [14], a tour constructed using a space filling curve is within a factor 1.25 from the optimum in the worst case if a uniform distribution of the points is used.

Several metaheuristic approaches have been proposed in recent twenty years. Gendreau et al. [37, 38], Golden et al. [40], and Laporte et al. [48] proposed metaheuristics approaches for VRP and analyzed their performances. Toth and Vigo [89] described a new variant of tabu search approaches, which they call granular tabu search. For other VRP algorithms, see the bibliographies by Laporte [49] and by Laporte and Osman [50].

Many applications of VRP are described in the practical problems: soft-drink distribution [71], oil industry [36], bulk sugar delivery [91], brewing industry [31], food distribution [20], transportation of live animals [65, 81], and so on. According to Toth and Vigo [87], in the

large number of real-world applications both in North America and in Europe, the use of computerized procedures for the distribution process planning produces substantial savings from 5% to 20% in the grobal transportation costs.

### 1.4.2 Algorithms for SDVRP

We next introduce algorithms for SDVRP. Dror and Trudeau [28, 29] first proposed an algorithm for SDVRP. The algorithm consists of two stages. In the first stage, an initial solution to VRP is constructed, and in the second stage, a split procedure which splits a demand for a customer between two vehicles is conducted iteratively in order to reduce the total travel cost. They observe that, when customer demand is small relative to vehicle capacity, there are almost no split deliveries, while when customer demand is very large (e.g., from 0.7 to 0.9 times vehicle capacity), split deliveries occur and the total travel cost is reduced (e.g., the average savings of 11% over a VRP solution). The detail of the algorithm will be described in Chapter 2.

Frizzell and Giffin [34] developed a construction heuristic for SDVRP on a grid network. Furthermore they [35] studied SDVRP with time windows on a grid network. Dror et al. [30] formulated SDVRP as an integer linear program with several new classes of valid constraints, and developed a constraint relaxation algorithm using branch and bound for solving SDVRP exactly. Belenguer et al. [8] proposed a lower bound for SDVRP based on a polyhedral study of the problem. They introduced a new family of valid inequalities and proposed a cutting-plane algorithm for generating a lower bound to SDVRP. Archetti et al. [4] proposed a tabu search algorithm for SDVRP. Recent dissertations by Liu [54] and Nowak [64] modeled variants of the SDVRP, proposed solution procedures, and reported computational results. Chen and Wasil [17] proposed a new heuristic that combines a mixed integer program and a record-to-record travel algorithm. Their heuristic generally performs better than tabu search.

Several applications of SDVRP are also described in the practical problems. Mullaseril et al. applied Dror and Trudeau's algorithm [28, 29] for SDVRP to a problem of livestock feed distribution [58]. Other applications of SDVRP are newspaper logistics [84], routing of helicopters [80], and so on.

### 1.4.3 Algorithms for variants of PDP

The pickup and delivery problem with time windows (PDPTW) has been widely studied recently [9, 51, 63, 68, 74, 77]. Nanry and Barnes [63] first presented a metaheuristic algorithm that incorporates a tabu search method for PDPTW. Li and Lim [51] adopted simulated annealing and tabu search method to PDPTW, and analyzed its performance by using instances that are generated from instances for VRPTW by Solomon [82].

Shang and Cuff [78] first introduced an operation of transfers, where requests can be

exchanged between vehicles. Minic and Laporte [56] proposed an algorithm for the pickup and delivery problem with time windows and transfers. Their algorithm consists of two phases; the first is a construction phase based on cheapest insertions, the second is an improvement phase by reinserting requests. They investigated the usefulness of transfers and showed that transshipment points reduce the total travel cost when requests are uniformly generated in the plane. Furthermore they showed that transshipment points are useful in clustered instances and their usefulness increased when the cluster size becomes smaller. Cortés et al. [19] have proposed a mixed integer programming (MILP) model for the pickup and delivery problem with time windows and transfers. The authors have implemented a branch-and-cut algorithm, and solved very small instances with up to eight customers and one transshipment point.

## 1.5 Analyses on the optimal cost between problems

As described in Section 1.1, the maximum saving that can be achieved by making a change to the routing rules and facilities is usually large, and the information about the maximum possible cost saving by each types of constraints on routing and facilities is helpful for companies before conducting changes actually. In this section, we introduce theoretical analyses on how the maximum travel cost can be saved by dropping constraints. Some experimental results have been shown how much travel cost is saved by dropping constraints in PDP type problems [29, 56]. We focus on theoretical analyses afterward in this section.

Given an instance $I$ and problem PRB, let $p$ be the number of requests that need to be served, and let $opt_{\mathrm{PRB}}(I)$ denote the optimal cost to PRB with instance $I$. For some $\mathrm{PRB}_1$ and $\mathrm{PRB}_2$, the maximal cost that can be saved by regarding an instance to $\mathrm{PRB}_1$ as that to $\mathrm{PRB}_2$ has been studied.

Archetti et al. [5] analyzed travel cost that can be saved by allowing split deliveries to the standard VRP. They showed that $opt_{\mathrm{VRP}}(I) \leq 2opt_{\mathrm{SDVRP}}(I)$ holds for any instance $I$. They proved the bound by converting a solution to SDVRP to a solution to VRP, and showing the increased travel cost is less than or equal to $opt_{\mathrm{SDVRP}}(I)$. They furthermore gave an instance where the bound is tight. They also study a variant of VRP such that a demand of a customer may be larger than the vehicle capacity, while each customer has to be visited a minimum number of times. They showed that travel cost saved by allowing more than the minimum number of required visits is again at most 50%.

Charikar et al. [16] analyzed the maximum travel cost that can be saved by introducing a transshipment point to $k$-delivery traveling salesman problem. The $k$-delivery problem is as follows. Given $n$ identical objects that are placed at arbitrary initial locations, and required to be delivered to $n$ target locations, the problem asks to find a route with the minimum cost where a vehicle delivers at most $k$ items at a time. They showed that the ratio of distance

traveled by an optimal route with no transshipments versus a route with transshipments is bounded by 4.

Arkin et al. [6] considered a delivery problem on a network where nodes have supplies or demands, and the total supply is equal to the total demand. They assume that arcs satisfy the Triangle Inequality, and a vehicle has infinite capacity. They consider certain restrictions on routes, and compare the quality of solutions of the unrestricted problem to that of the restricted one. They especially considered a delivery problem with a restriction that all pickup actions must be made before any delivery actions, which corresponds to a constraint of consecutive pickups and deliveries (constraint (e) in Section 1.3). They showed that the ratio of the restricted optimal solution to the unrestricted optimal solution is 2.

## 1.6 Main contribution and outline of the thesis

In this thesis, we make two types of contributions.

One contribution is proposing a fast construction algorithm to find a cheaper routing schedule, which is one approach to decrease physical distribution cost. We consider the discrete split delivery vehicle routing problem (DSDVRP), which is introduced in Section 1.3, and propose a fast algorithm that constructs routes one by one without any improvement procedures to the DSDVRP. The algorithm generates routes by a dynamic programming based on an elaborate route evaluation function that estimates the total travel cost required to service all the remaining items by vehicles.

In Chapter 2, we make preparations to show the algorithm for DSDVRP. We first define problems VRP and SDVRP, and describe standard construction algorithms for VRP; the saving algorithm, the insertion algorithm, and the sweep algorithm. For SDVRP, we introduce Dror and Trudeau's heuristic algorithm [28, 29]. Those algorithms are used to compare with our algorithm in Chapter 3.

In Chapter 3, we explain the detail of the construction algorithm for DSDVRP [60]. We furthermore conduct computational experiments for real-world instances and show that our algorithm is practically efficient and fast comparing to representative heuristics for VRP and SDVRP. Our approach to estimate the total travel cost of resulting items accurately would be applied to other types of routing problems in order to produce good quality solutions with short computation time. Furthermore mixing several improvement methods including sophisticated metaheuristics with our construction algorithm could produce good solutions with less travel cost.

The other contribution is analyzing the maximum possible cost that can be saved by relaxing some constraints. As described in Section 1.1, it is desirable to know the possible cost saving by each type of constraints on routing without conducting accurate estimations of their performances.

Fig. 1.4 shows factors of upper and lower bounds on the maximum cost that can be saved by regarding an instance to problem A as that to problem B, where A (resp., B) corresponds to the problem that is located on the head (resp., tail) of each arc in Fig. 1.4. A value on the left (resp., right) side of each arc indicates an upper bound (resp., lower bound) between two problems of head and tail of the arc. Note that VRP is a problem of PDPC with special instances. Factor $O(p)$ on the maximum ratio on the optimal cost of PDPC to that of PDPCMT can be easily confirmed. Thus one of the our final goal in this thesis is to find upper/lower bounds between every pair of the problems in Fig 1.4 in terms of common input parameters such as the number $p$ of requests.

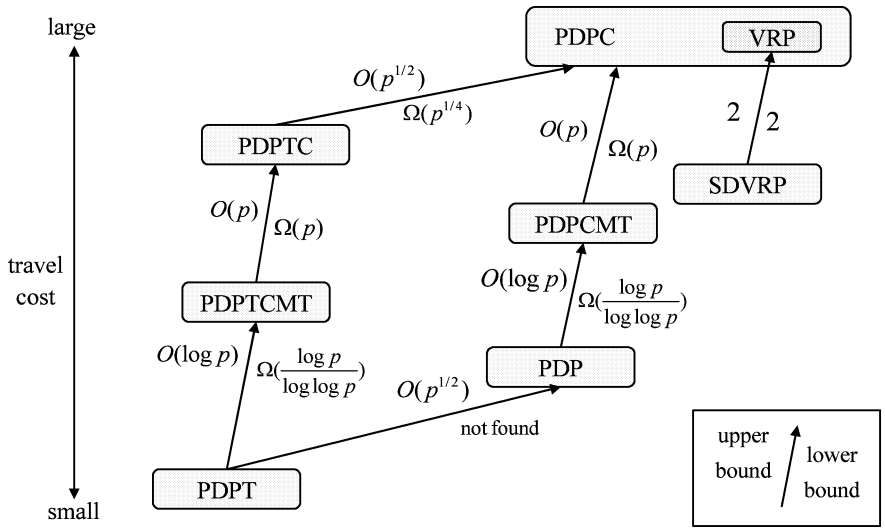



Figure 1.4: Upper and lower bounds on the maximum ratio on the optimal cost, where "not found" between problems PDPT and PDP means that no bounds have been found between them.

In this thesis, we will make the following analyses. In Chapter 4, we study the maximum cost that can be saved by introducing a transshipment point to PDP [62]. We will show that

$$opt_{\mathrm{PDP}}(I) \leq (6\lceil p^{1/2} \rceil + 1) \cdot opt_{\mathrm{PDPT}}(I)$$

holds for any instance $I$ with $p$ requests and one transshipment point if each vehicle can visit the transshipment point at most once. The inequality is shown by presenting a polynomial time algorithm that converts a solution to PDPT to a solution to PDP, and by showing that the travel cost of the constructed solution is bounded by the ratio. We then show that if each vehicle can visit the transshipment point any number of times, then it holds

$$opt_{\mathrm{PDP}}(I) \leq (6\lceil p^{1/2} \rceil + 2) \cdot opt_{\mathrm{PDPT}}(I).$$

Furthermore we see that the bound is valid for the ratio of $opt_{\mathrm{PDPC}}(I)$ to $opt_{\mathrm{PDPTC}}(I)$, and we present an instance $I'$ that satisfies $opt_{\mathrm{PDPC}}(I') = p^{1/4} \cdot opt_{\mathrm{PDPTC}}(I')$.

In Chapter 5, we examine how the least travel cost can be increased by the requirement to PDP that no vehicle which has begun a delivery action is allowed to take pickup actions until all of the loads on the vehicle are delivered [61]. We will show that the maximum ratio of the optimal value of PDPCMT to that of PDP over all instances with $p$ requests is presented by

$$opt_{\mathrm{PDPCMT}}(I) \leq \lceil \log_2 2p \rceil \cdot opt_{\mathrm{PDP}}(I).$$

The analysis for the upper bound can be easily extended to a case with a transshipment point. We will show that the maximum ratio of the optimal value of PDPTCMT to that of PDPT over all instances with $p$ requests is given by

$$opt_{\mathrm{PDPTCMT}}(I) \leq \lceil \log_2 4p \rceil \cdot opt_{\mathrm{PDPT}}(I).$$

We furthermore present an instance $I$ that gives a lower bound that satisfies

$$opt_{\mathrm{PDPCMT}}(I) \geq \frac{\log_2(p+1)}{4 \log_2 \log_2(p+1)} \cdot opt_{\mathrm{PDP}}(I).$$

The lower bound holds for the case with a transshipment point by regarding that the transshipment point is located at a sufficiently far location such that using the transshipment point does not contribute to reduce the total cost. Thus there is an instance $I$ that satisfies

$$opt_{\mathrm{PDPTCMT}}(I) \geq \frac{\log_2(p+1)}{4 \log_2 \log_2(p+1)} \cdot opt_{\mathrm{PDPT}}(I).$$

In order to show the lower bound, we first introduce a partition problem that asks to find a partition of vertices on a complete binary tree with the minimum cost, and present a lower bound on the partition problem. We analyze a lower bound on travel cost of a PDPCMT solution by considering travel cost that does not depend on sequences on trips but are derived from travel cost between pickup points and delivery points of requests in the trips. A lower bound of the travel cost is obtained by using the result of the lower bound for the partition problem.

Finally in Chapter 6, we make concluding remarks.

# Chapter 2

# Heuristic Algorithms for VRP and SDVRP

## 2.1 Introduction

In this chapter, we first formulate the vehicle routing problem (VRP) and the split delivery vehicle routing problem (SDVRP). VRP is the most common and well studied among a number of routing problems and it has been a subject of intensive research focused mainly on heuristic and metaheuristics approaches. We next explain three classic heuristic algorithms for VRP; the saving method, the sweep method, and the insertion method. Furthermore we review a local search and some representative neighborhoods; the $\lambda$-opt neighborhood, the or-opt neighborhood, and the cross exchange neighborhood. We finally review Dror and Trudeau's heuristic algorithm for SDVRP. Those algorithms are used to compare solution quality with that of our algorithm for DSDVRP in Chapter 3.

## 2.2 Problem definition

This section formulates problem VRP. We are given a vertex set $V = \{0, 1, \ldots, n\}$, where $0 \in V$ denotes the depot while the other vertices are *customers*. We can regard vertex 0 as a pickup point of all requests, and each customer as a delivery point in PDP. The number of requests in PDP is equal to $|V| - 1$, i.e., the number of customers. The following inputs are associated with each customer $j \in V - \{0\}$.

- $q(j)$ : Quantity of goods required at customer $j \in V - \{0\}$. Denote $q(V') = \sum_{j \in V'} q(j)$ for each subset $V' \subseteq V - \{0\}$. We assume that quantity $q(j)$ for each customer $j \in V - \{0\}$ is a positive real number.

- $\mathcal{R}$ : A set of vehicles available. We suppose that the vehicles are homogeneous, and

there are a sufficiently large number of vehicles available to serve all customer demands. We denote the vehicle capacity by $c$, and suppose that $c$ is a positive real number.

- $d(j, j')$ : Distance from customer (or depot) $j$ to customer (or depot) $j'$ for $j, j' \in V$. Distance $d(j, j')$ is a nonnegative real number, and in general asymmetric, i.e., $d(j, j') \neq d(j', j)$ may hold.

In this chapter, a *route* $\lambda$ is a sequence $j_1, j_2, \ldots, j_u$ of customers, and is denoted by $[j_1, j_2, \ldots, j_u]$. Its *travel cost* $d(\lambda)$ is defined to be

$$d(\lambda) = d(0, j_1) + \sum_{1 \leq i \leq u-1} d(j_i, j_{i+1}) + d(j_u, 0).$$

For simplicity, we may treat a route $\lambda$ as an ordered subset of $V$; For example, $q(\lambda)$ means the total quantity of goods for customers in route $\lambda$. A vehicle visits the customers in a route in the order of the sequence of the route.

Problem VRP is formulated as follows:

$$\begin{aligned}
\text{minimize} \quad & \sum_{\nu \in \mathcal{R}} d(\lambda_\nu) && (2.1)\\
\text{subject to} \quad & \cup_{\nu \in \mathcal{R}} \lambda_\nu = V - \{0\} && (2.2)\\
& \lambda_\nu \cap \lambda_{\nu'} = \emptyset \ \text{ for all distinct } \nu, \nu' \in \mathcal{R} && (2.3)\\
& q(\lambda_\nu) \leq c \ \text{ for all } \nu \in \mathcal{R}. && (2.4)
\end{aligned}$$

In the above formulation, $\lambda_\nu$ is a route that is served by vehicle $\nu \in \mathcal{R}$. If vehicle $\nu$ is not utilized, then we let $\lambda_\nu = \emptyset$ and $d(\lambda) = 0$. Constraint (2.2) indicates that all customers in $V - \{0\}$ must be visited by any of the vehicles, and constraint (2.3) indicates that every customer is included in exactly one route. Constraint (2.4) means capacity constraint. The objective is to minimize the total travel cost as shown by (2.1).

We next define the split delivery vehicle routing problem (SDVRP). In SDVRP, each vehicle can visit a customer more than once. Thus we give the following additional definition.

- $q(\lambda_\nu, j)$ : Quantity of goods served to customer $j \in V - \{0\}$ on route $\lambda_\nu$.

By using the notation $q(\lambda_\nu, j)$, problem SDVRP is formulated as follows:

$$\begin{aligned}
\text{minimize} \quad & \sum_{\nu \in \mathcal{R}} d(\lambda_\nu) && (2.5)\\
\text{subject to} \quad & \cup_{\nu \in \mathcal{R}} \lambda_\nu = V - \{0\} && (2.6)\\
& \sum_{\nu \in \mathcal{R}} q(\lambda_\nu, j) = q(j) \ \text{ for all } j \in V - \{0\}. && (2.7)\\
& q(\lambda_\nu) \leq c \ \text{ for all } \nu \in \mathcal{R}. && (2.8)
\end{aligned}$$

Constraint (2.7) indicates that the total quantity of demand that is served for customer $j$ by all vehicles is equal to demand $q(j)$ for customer $j$.

## 2.3 Classical construction algorithms for VRP

In this section, we describe representative classical heuristic algorithms for VRP; the saving method, the sweep method, and the insertion method. In Chapter 3, solution quality of our algorithm for the discrete split delivery vehicle routing problem will be compared with one by the first two algorithms.

### 2.3.1 Saving algorithm

The saving method was proposed for VRP with no time windows by Clarke and Wright [18]. It begins with a solution in which each vehicle visits exactly one customer, that is, each vehicle goes to a customer from the depot, and after serving demand for the customer, it returns to the depot. Next the following merging procedure is repeated. Let $s_{ij}$ be a cost saving that is achieved by merging two routes, where the last customer of one route is customer $i$ and the first customer of the other route is customer $j$. Then cost $s_{ij}$ is represented by

$$s_{ij} = d(i,0) + d(0,j) - d(i,j).$$

The algorithm selects an edge $(i,j)$ such that cost $s_{ij}$ is the maximum under the restriction that the total quantity of demand that are served on the two routes does not exceed capacity $c$, and then merges them. The saving algorithm is described as follows.

**Algorithm** Saving Algorithm
**Input:** A set $V$ of vertices, where $0 \in V$ is the depot and $v \in V - \{0\}$ is a customer with demand $q(v)$, distance $d$ between two vertices in $V$, and a set $\mathcal{R}$ of vehicles with capacity $c$.
**Output:** A set $\Lambda = \{\lambda_1, \ldots, \lambda_r\}$ of routes.

**1:** Generate routes $\lambda_j = [j]$ for each customer $j \in V - \{0\}$;
**2:** Compute savings $s_{ij} = d(i,0) + d(0,j) - d(i,j)$ for each customer $i$ and customer $j$, and order the savings in a descending order;
**3:** **for** Start from the top of the savings list **do**
**4:** Given a saving $s_{ij}$, determine whether there exist two routes, one containing edge $(0,j)$ and the other containing edge $(i,0)$ such that the total quantity of demands that are served on the two routes does not exceed capacity $c$;
**5:** **if** Such two routes exist **then**
**6:** Merge the two routes by deleting edge $(i,0)$ and edge $(0,j)$ and introducing edge $(i,j)$
**7:** **end** /* if */

**8: end**. /* for */

Fig 2.1 illustrates an example of an operation that merges two routes. Note that a sequence of one route is traversed by the merge.

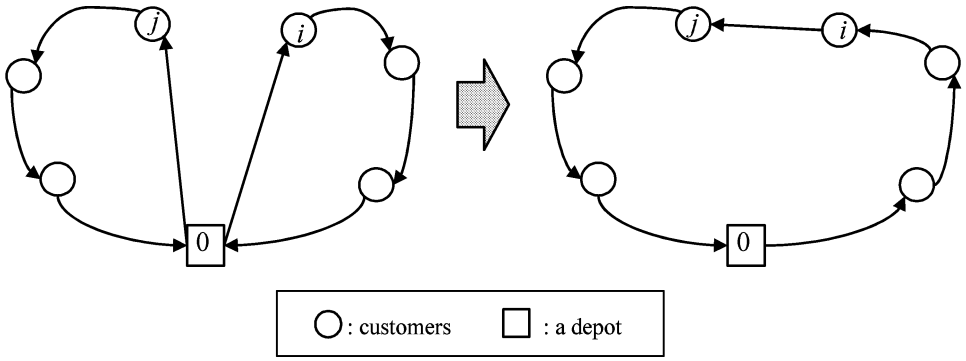



Figure 2.1: An example of an operation that merges two routes by the saving algorithm.

Paessens [67] proposed the following savings function;

$$s_{ij} = d(i,0) + d(0,j) - g \cdot d(i,j) + f \cdot |d(i,0) - d(0,j)|,$$

where $0 < g \leq 3$ and $0 \leq f \leq 1$ are parameters. They showed that an example of good parameter values is $g = 1.4$ and $f = 0.5$. Furthermore Paessens discussed some techniques to reduce storage requirements.

### 2.3.2 Sweep algorithm

Gillet and Miller [39] proposed an algorithm for the Euclidean VRP which is based on the geographical direction of customers. They assumed that all vehicles have the same capacity. Positions of customers are given in polar coordinates. The depot is supposed to locate in the center. Vertices are ordered in an ascending or descending order of their arguments. If two vertices have the same argument, then the vertex which has the smallest argument comes first. The algorithm divides a set of customers in the order so that the total quantity of requests for each subset that is served by one vehicle does not exceed the capacity.

Assume that we have ordered a set of customers, and renumbered them according to the order (note that vertex 0 is the depot). Let $V'$ be a set of customers that are assigned to any of routes. Given two sequences $\sigma_1$ and $\sigma_2$ of vertices, we denote by $\sigma_1 \oplus \sigma_2$ a sequence that follows sequences $\sigma_1$ and $\sigma_2$ in this order. The sweep algorithm is described as follows.

**Algorithm** SWEEP ALGORITHM

**Input:** A set $V$ of vertices, where $0 \in V$ is the depot and $v \in V - \{0\}$ is a customer with demand $q(v)$, where customers are ordered and renumbered by an ascending or descending order of arguments, and a set $\mathcal{R}$ of vehicles with capacity $c$.

**Output:** A set $\Lambda = \{\lambda_1, \ldots, \lambda_r\}$ of routes.

**1:** $V' := \emptyset$ and $i := 1$;
**2:** Start with the depot for the first vehicle, i.e., $\lambda_1 = \emptyset$;
**3:** **while** $V' \neq V - \{0\}$ **do**
**4:** Select a customer $j \in V - V' - \{0\}$ with the smallest index;
**5:** **if** $q(\lambda_i) + q(j) \leq c$ **then**
**6:** Assign customer $j$ to the current vehicle, i.e., $\lambda_i := \lambda_i \oplus [j]$;
**7:** $V' := V' \cup \{k\}$
**8:** **else**
**9:** The current vehicle returns to the depot;
**10:** Prepare a new vehicle, i.e., $i := i + 1$ and $\lambda_i = \emptyset$
**11:** **end** /* if */
**12:** **end**. /* while */

Fig. 2.2 illustrates an example of a solution with three routes constructed by the sweep algorithm. Each number besides a vertex represents the order of an assignment to a vehicle.

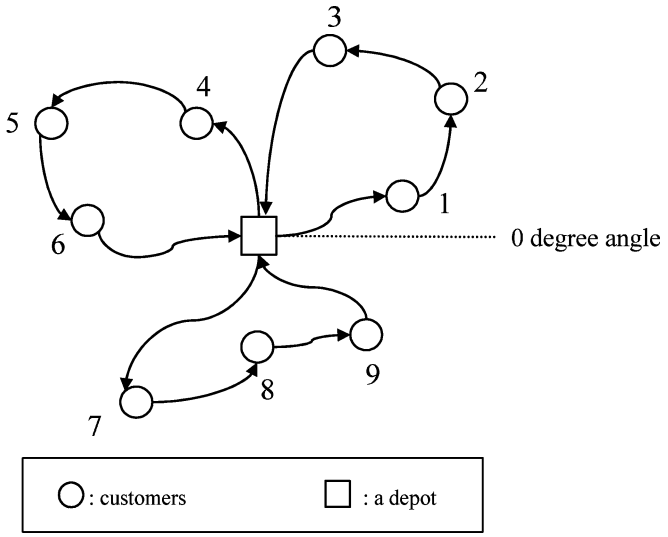



Figure 2.2: An example of a solution obtained by the sweep algorithm.

### 2.3.3 Insertion algorithm

The insertion algorithm was first proposed and analyzed for the traveling salesman problem by Rosenkrantz et al. [75], and it was applied to the vehicle routing problem with time windows by Solomon [82].

Given a constant $r$, it first selects $r$ customers, which are called *seed customers*. Then it constructs a solution with $r$ routes in which each vehicle visits exactly one seed customer, that is, each vehicle goes to a seed customer from the depot, and after serving demand for the seed customer, it returns to the depot. Next the following inserting procedure is repeated. Let $k$ be an unrouted customer, and let $i$ and $j$ ($i \neq j$) be two consecutive vertices (customers or the depot) that belong to the same route. We denote by $s_{ijk}$ a cost that is achieved by inserting customer $k$ between vertices $i$ and $j$. Then cost $s_{ijk}$ is represented by

$$s_{ijk} = d(i,k) + d(k,j) - d(i,j).$$

Given an unrouted customer $k$, the algorithm determines vertices $i$ and $j$ such that cost $s_{ijk}$ is the minimum under the restriction that the total quantity of demand that is served on a route does not exceed capacity $c$, and insert $k$ between vertices $i$ and $j$. Let $V'$ be a set of customers that belong to any of constructed routes. The insertion algorithm is described as follows.

**Algorithm** INSERTION ALGORITHM
**Input:** A set $V$ of vertices, where $0 \in V$ is the depot and $v \in V - \{0\}$ is a customer with demand $q(v)$, distance $d$ between two vertices in $V$, a set $\mathcal{R}$ of vehicles with capacity $c$, and a constant $r$.
**Output:** A set $\Lambda = \{\lambda_1, \ldots, \lambda_r\}$ of routes.

**1:** Select $r$ seed customers, and generate $r$ routes each of which includes exactly one seed customer;
**2:** **while** $V' \neq V - \{0\}$ **do**
**3:** Select a customer $k \in V - V' - \{0\}$;
**4:** Compute costs $s_{ijk} = d(i,k) + d(k,j) - d(i,j)$ for all consecutive vertices $i$ and $j$ on the current routes;
**5:** Select vertices $i$ and $j$ such that cost $s_{ijk}$ is the minimum under the restriction that the total quantity of demand after inserting customer $k$ does not exceed capacity $c$;
**6:** **if** Such vertices $i$ and $j$ exist **then**
**7:** Insert customer $k$ between vertices $i$ and $j$
**8:** **end**; /* if */
**9:** $V' := V' \cup \{k\}$
**10:** **end**. /* while */

There exist variations in selecting seed customers at Line 1 and a customer $k$ at Line 3. When implementing the insertion algorithm in Chapter 3, we select seed customers randomly, and select a customer $k$ that is the farthest from the depot in $V - \{0\} - V'$. Campbell and Savelsberg [15] discussed an efficient implementation of the insertion algorithm for vehicle routing problems with complicated constraints.

## 2.4 Local search

In this section, we review local search that improves initial solutions generated by construction algorithms, some of which are introduced in the previous section.

Local search starts from an initial solution, and iteratively replaces it with a better solution in its *neighborhood*, which is a set of solutions that are obtained from the current solution by perturbing slightly [1, 43, 69, 92, 93]. Local search terminates when it reaches a solution that has no better solutions in the neighborhood, or a time bound is elapsed. There exist several problems where local search has been applied as follows.

- the travelling salesman problem [21, 52, 53], in which a solution is a cycle containing all nodes of the graph and the target is to minimize the total length of the cycle;
- the vehicle routing problem, the pickup and delivery problem and their variants [4, 12, 13, 37, 40, 85, 86, 89];
- the vertex cover problem [73, 79], in which a solution is a vertex cover of a graph, and the target is to find a solution with a minimal number of vertices;
- the boolean satisfiability problem [33, 41], in which a candidate solution is a truth assignment, and the target is to maximize the number of clauses satisfied by the assignment; in this case, the final solution is of use only if it satisfies all clauses.
- the nurse scheduling problem [2, 7, 27] where a solution is an assignment of nurses to shifts which satisfies all established constraints.

Definitions of neighborhoods depend on problems, and are essential parts in designing a local search algorithm. As an example, the neighborhood of a vertex cover is another vertex only differing by one vertex. For boolean satisfiability, neighborhoods of a truth assignment are usually truth assignments only differing from it by an evaluation of a variable.

Local optimization with neighborhoods that involve changing up to $k$ components of a solution is often referred to as *k-opt*. A locally optimal solution with a large neighborhood usually produces better solutions than that with a smaller neighborhood, while the neighborhood search with a larger neighborhood often takes more computation time.

There exist two types of move strategies in local search, one is the *first admissible move strategy* and the other is the *best admissible move strategy*. The first admissible move strategy scans solutions in neighborhoods according to a prespecified order, and immediately accepts a solution that improves the current solution first as the next solution. The best admissible move strategy selects the best solution in neighborhoods as the next solution. In many cases, first admissible move strategy is applied since best admissible move strategy requires much computation time.

Metaheuristics are the most enhanced method in approximate optimization technique of the last thirty decades. Tabu search is a representative metaheuristic method. Many high quality solutions for VRP by tabu search have been reported [42, 85, 86, 89]. Metaheuristics methods usually produce better solutions than those by construction algorithms, however they consume much computation time and sometimes require finely tuned parameters. Thus we attempt to produce good solutions by a construction algorithm in Chapter 3.

### 2.4.1 Neighborhoods for VRP

In this section, we introduce three representative neighborhoods for the traveling salesman problem and the vehicle routing problem. For more information, see [13, 42, 47, 57].

(i) $\lambda$-opt neighborhood : The $\lambda$-opt neighborhood was proposed by Lin [52] for the traveling salesman problem, where $\lambda$ is a prescribed integer. Given a route, it removes $\lambda$ edges of a route, and connects the resulting $\lambda$ segments so that a route is constructed. The 2-opt neighborhood is probably the most basic and widely used neighborhood in TSP. Given a route, it selects two edges $(i, m)$ and $(j, k)$ from the route such that all indices $i, m, j$ and $k$ are distinct, and appear in this order in the route. The 2-opt replaces these two edges by edges $(i, j)$ and $(m, k)$ if this change decreases travel cost of the current route. This simple heuristics performs well on Euclidean instances, e.g., the well-known TSPLIB [72]. Local search by 2-opt needs a subquadratic number of improving steps until it reaches a local optimum, and the constructed solution lies within a few percentage points of the global optimum [46]. Fig 2.3 illustrates an example of the 2-opt neighborhood operation. In Fig 2.3, two edges $(i, m)$ and $(j, k)$ are removed, and two edges $(i, j)$ and $(m, k)$ are inserted.

(ii) Or-opt neighborhood : The or-opt neighborhood, which was proposed by Or [66] in 1976, is an $O(|V|^2)$ exchange operation that produces solutions nearly as good as 3-opt. It first removes edges of three consecutive vertices, and consider all possible reinsertions. Fig 2.4 illustrates an example of the or-opt neighborhood operation. In Fig 2.4, three consecutive vertices from $i$ to $j$ are removed, and inserted between vertices $m$ and $k$.

(iii) Cross exchange neighborhood : Given a constant $L^{cross}$, the cross exchange neighborhood [86] is defined to be a set of solutions that are obtained by exchanging two paths of

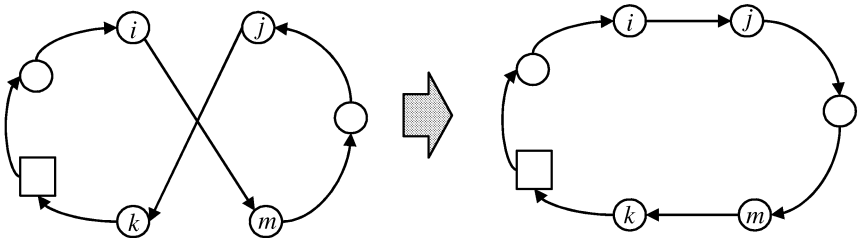



Figure 2.3: An example of the 2-opt neighborhood operation.

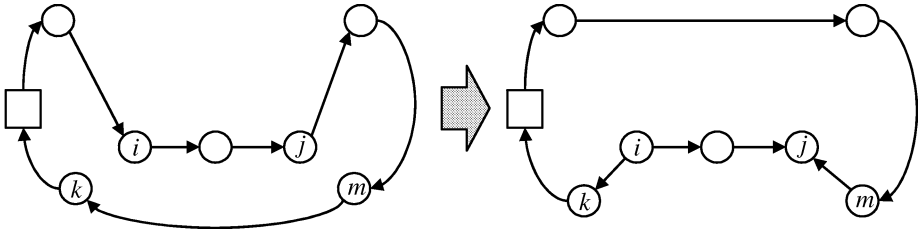



Figure 2.4: An example of the or-opt neighborhood operation.

length, i.e., the number of edges, at most $L^{cross}$ between two routes. This neighborhood can be regarded as a subset of the 4-opt neighborhood. The cross exchange neighborhood is an extension where the exchanged paths can be inserted with the reverse order [12]. Fig 2.5 illustrates an example of the cross exchange neighborhood operation.

## 2.5 Dror and Trudeau's algorithm for SDVRP

In this section, we describe the most well-known algorithm for SDVRP by Dror and Trudeau [28, 29]. They consider the case where demand of each customer is not beyond a vehicle capacity. Their algorithm mainly consists of two phases. Firstly an initial VRP solution is constructed, and secondary a local search algorithm that includes a splitting operation of a customer demand is operated. The local search is composed of the following two procedures.

(i) $k$-split interchange : This procedure removes a customer from all the routes that include the customer, and reinsertions the customer into routes by admitting splitting the customer demand. We consider a case where demand $q(i)$ of customer $i$ is split below.

**1** Remove customer $i$ from all the routes where customer $i$ is served.

**2** Consider all subsets $\Lambda'$ of $k$ routes such that the total residual capacity of routes in $\Lambda'$ is larger than or equal to demand $q(i)$ of customer $i$. For each such subset $\Lambda'$,

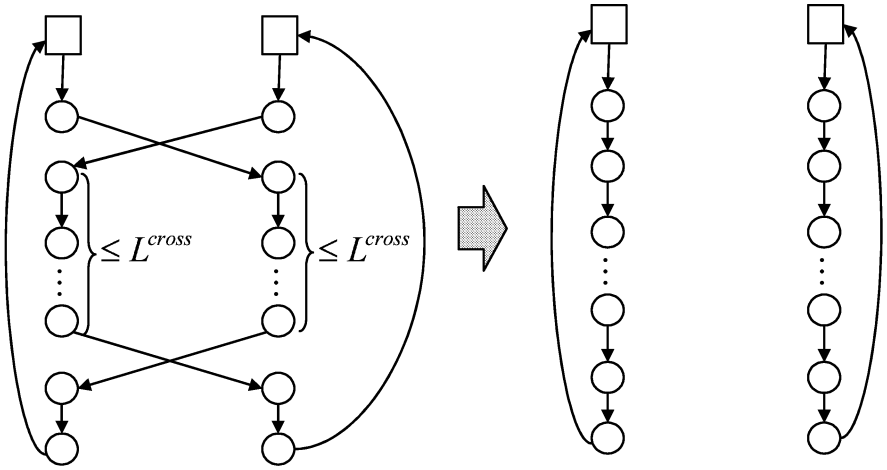



Figure 2.5: An example of the cross exchange neighborhood operation.

compute the total insertion cost of customer $i$ into all routes in $\Lambda'$ by splitting the customer demand. The order to visit customer $i$ in each route in $\Lambda'$ is determined so that the insertion cost is minimized. Choose a subset $\Lambda'$ that leads to the minimum insertion cost, and insert $i$ into all routes in $\Lambda'$.

(ii) Route addition : This procedure eliminates a split delivery for a customer to reduce the total travel cost. Suppose that customer $i$ is split, and appears in at least two routes $\lambda_1$ and $\lambda_2$. We eliminate the split of demand $q(i)$ of customer $i$ on these two routes, and create a new route by checking 9 different possible route configurations with the following properties.

**1** Preserve the four principle route segments on routes $\lambda_1$ and $\lambda_2$ (from the depot to the customer preceding $i$ and from the customer succeeding $i$ to the depot).

**2** Create three routes considering all the possible combinations among the principle route segments and customer $i$, and choose the best one (see [28, 29] for detail).

The route addition procedure among three different routes produces 19 possible combinations. In case a customer is split between more than three routes, the route addition procedure examines each instance of two and three routes that visit the customer and analyses 9 and 19 configurations, respectively.

The main algorithm consists of five subroutines. The first subroutes constructs an initial VRP solution by a variant by Bodin et al. [10] of the well-known saving algorithm. The second subroutine is a local search using the cross exchange neighborhoods with $L^{cross} = 1$ and $L^{cross} = 2$. The third one is a local search with the 2-opt. The fourth is the $k$-split

interchange procedure, and the fifth subroutine is the route addition procedure. The main algorithm is described as follows.

**Algorithm** Dror and Trudeau's Algorithm for SDVRP
**Input:** A set $V$ of vertices, where $0 \in V$ is the depot and $v \in V - \{0\}$ is a customer with demand $q(v)$, distance $d$ between two vertices in $V$, and a set $\mathcal{R}$ of vehicles with capacity $c$.
**Output:** A set $\Lambda = \{\lambda_1, \ldots, \lambda_r\}$ of routes.

**1:** Construct an initial VRP solution;
**2:** Execute a local search with the cross exchange neighborhood and the 2-opt;
**3:** Execute $k$-split interchange procedures for all customers;
**4:** Execute route addition procedures for all customers;
**5:** **if** A improvement occurs for at least one customer at Line 5 **then**
**6:** goto Line 4
**7:** **end**; /* if */
**8:** **if** A improvement occurs for at least one customer at Line 4 **then**
**9:** goto Line 2
**10:** **end**. /* if */

# Chapter 3

# DP-based Heuristic Algorithm for the Discrete Split Delivery Vehicle Routing Problem

## 3.1 Introduction

In this chapter, we consider the discrete split delivery vehicle routing problem (DSDVRP). In DSDVRP, each customer may be visited more than once, however, loads for a customer consists of a set of items, each of which cannot be split, where items may have different sizes. For this problem, we propose a DP-based heuristic algorithm that constructs routes one by one without any improvement procedures. The algorithm generates each route by dynamic programming based on an elaborate route evaluation function that estimates the total travel cost that is required to service all the remaining items by vehicles. We show that our algorithm is practically efficient and fast comparing to representative heuristic algorithms for VRP and SDVRP that are introduced in the previous chapter.

## 3.2 Problem definition

This sections gives some definitions, and formulates the discrete split delivery vehicle routing problem.

Let $V = \{0, 1, \ldots, n\}$ be a set of vertices, where $0 \in V$ denotes the depot while the other vertices are customers. We call an element of goods which is delivered to a customer an *item*. Let $M$ stand for a set of all items over all customers in $V - \{0\}$, and $m$ denote $|M|$. The following inputs are associated with each customer $j \in V - \{0\}$ and each item $k \in M$.

- $M_j$ : A set of items which are required to be delivered from depot 0 to customer $j \in V - \{0\}$ (hence $M = \bigcup_j M_j$ and $M_j \cap M_{j'} = \emptyset$ for $j \neq j'$).

- $q(k)$ : Quantity of item $k \in M$. Denote $q(M') = \sum_{k \in M'} q(k)$ for each subset $M' \subseteq M$. We assume that quantity $q(k)$ of each item $k \in M$ are positive integers. The assumption of positive integers is required for our algorithm that is on a dynamic programming in Section 3.3.2.

- $\mathcal{R}$ : A set of vehicles available. We suppose that the vehicles are homogeneous, and there are a sufficiently large number of vehicles available to service all items, e.g., $|\mathcal{R}| \geq |M|$ holds. We assume that vehicle capacity $c$ is a positive integer.

- $d(j, j')$ : Distance from customer (or depot) $j$ to customer (or depot) $j'$ for $j, j' \in V$. Distance $d(j, j')$ is a nonnegative real number, and in general asymmetric, i.e., $d(j, j') \neq d(j', j)$ may hold. Distance $d(j, j')$ may not satisfy the Triangle Inequality.

A route $\lambda$ is a sequence $k_1, k_2, \ldots, k_u$ of items, and denoted by $[k_1, k_2, \ldots, k_u]$. Its travel cost $d(\lambda)$ is defined to be

$$d(\lambda) = d(0, k_1) + \sum_{1 \leq i \leq u-1} d(k_i, k_{i+1}) + d(k_u, 0).$$

For simplicity, we may treat a route $\lambda$ as an ordered subset of $M$; For example, $q(\lambda)$ means the total quantity of items in $\lambda$. A vehicle delivers the items in a route in the order of the sequence of the route.

Problem DSDVRP is formulated as follows:

$$\begin{aligned}
\text{minimize} \quad & \sum_{\nu \in \mathcal{R}} d(\lambda_\nu) && (3.1)\\
\text{subject to} \quad & \cup_{\nu \in \mathcal{R}} \lambda_\nu = M && (3.2)\\
& \lambda_\nu \cap \lambda_{\nu'} = \emptyset \ \text{ for all distinct } \nu, \nu' \in \mathcal{R} && (3.3)\\
& q(\lambda_\nu) \leq c \ \text{ for all } \nu \in \mathcal{R}. && (3.4)
\end{aligned}$$

In the above formulation, $\lambda_\nu$ is a route serviced by vehicle $\nu \in \mathcal{R}$. If vehicle $\nu$ is not utilized, then we let $\lambda_\nu = \emptyset$ and $d(\lambda) = 0$. Constraint (3.2) indicates that all items in $M$ must be served by any of the vehicles, and constraint (3.3) indicates that every item is included in exactly one route. Note that each customer is allowed to be visited by more than one vehicle. The objective is to minimize the total travel cost as shown by (3.1). The features of the problem are that customers may have multiple items, and the total quantity $q(M_j)$ for customer $j$ can be beyond vehicle capacity $c$.

## 3.3 DP-based heuristic algorithm

This section describes our algorithm for DSDVRP. We attempt to design a fast algorithm for constructing a set of routes without improving the resulting routes. We first describe a main

algorithm and a procedure used in the main algorithm.

### 3.3.1 Main algorithm and procedure

We first describe an outline of our algorithm which we call DPRoute. Algorithm DPRoute iteratively generates routes one by one until all the items in $M$ are assigned to one of the generated routes.

Let $M' \subseteq M$ stand for a set of items that are not yet assigned to any of the generated routes at an iteration in DPRoute. For a set $M'$ of items, a route for a subset of $M'$ is generated by a procedure called FindRoute$_\varphi$, which chooses a route based on a function $\varphi$ that evaluates "cost" of a route (we discuss how to choose function $\varphi$ in Section 3.3.3). DPRoute has an input parameter $g \in (0, 1]$ which is used in procedure FindRoute$_\varphi$ to control the least quantity of items assigned to each route.

**Algorithm** DPRoute
**Input:** A set $V$ of vertices, distance $d$ between two vertices in $V$, a set $M$ of items, quantity $q$ of items, a set $\mathcal{R}$ of vehicles with capacity $c$, and a parameter $g \in (0, 1]$.
**Output:** A set $\Lambda = \{\lambda_1, \ldots, \lambda_r\}$ of routes.

**1:** Set $M' := M$, $\nu := 0$ and $\Lambda := \emptyset$;
**2:** **while** $M' \neq \emptyset$ **do**
**3:** $\nu := \nu + 1$;
**4:** $\lambda_\nu :=$ FindRoute$_\varphi$ ($M'$,$g$);
**5:** $\Lambda := \Lambda \cup \{\lambda_\nu\}$;
**6:** $M' := M' - \lambda_\nu$
**7:** **end**; /* while */
**8:** $r := \nu$ and $\lambda_\nu := \emptyset$ for $\nu = r + 1, \ldots, |\mathcal{R}|$.

Note that $r \leq |\mathcal{R}|$ holds from the assumption that $|\mathcal{R}| \geq |M|$ in Section 3.2.

Given a subset $M' \subseteq M$, FindRoute$_\varphi(M', g)$ constructs a route as follows. For each $h = 1, 2, \ldots, c$, we choose a route $J_h$ with $q(J_h) = h$ by a procedure, called ConstructRoute$_\varphi$ based on an evaluation function $\varphi$, which will be discussed in Section 3.3.2. Function $\varphi$ will be defined so that a route $\lambda$ with a smaller $\varphi(\lambda)$ becomes a member of a "good" set of routes to the problem. Choice of function $\varphi$ is crucial to our algorithm, and will be discussed in Section 3.3.3.

**Procedure** FindRoute$_\varphi(M', g)$
**Input:** A set $M'$ of items and a parameter $g \in (0, 1]$.
**Output:** A route $\lambda$.

**1:** Construct routes $J_h$ with $q(J_h) = h$ for $h = 1, 2, \ldots, c$ by ConstructRoute$_\varphi(M')$, where

we let $J_h = \emptyset$ if no such $J_h$ exists; Let $h^*$ be the maximum $h$ such that $J_h \neq \emptyset$; (Note that such $h$ necessarily exists since it hold $|M'| > 0$ and $1 \leq q(k) \leq c$ for $k \in M'$. )
**2:** Select a route $\lambda := \mathrm{argmin}\{\varphi(J_h) \mid \lfloor g \cdot c \rfloor \leq q(J_h) \leq h^*\}$ if $\lfloor g \cdot c \rfloor \leq h^*$, and $\lambda := J_{h^*}$ otherwise.

The parameter $g$ gives the least quantity $\lfloor g \cdot c \rfloor$ of items in route $\lambda$. For a larger $g \in (0, 1]$, FindRoute$_\varphi(M', g)$ tends to select a route of larger quantity, which may lead to a solution with a less number of vehicles than that by a small $g$.

### 3.3.2 Constructing routes by dynamic programming

This section describes procedure ConstructRoute$_\varphi(M')$ which constructs routes $J_h$ with $q(J_h) = h$ for all $h = 1, 2, \ldots, c$ based on dynamic programming. Let $m' = |M'|$. We first renumber indices $k = 1, 2, \ldots, m'$ of items in $M'$ as follows. Let $k = 1$ be an item in the farthest customer from the depot $0 \in V$. For $k \geq 2$, $k$ is an item in the nearest customer from the customer with item $k - 1$, where we break ties in items with the same travel cost by selecting an item of the maximum quantity among them. By renumbering indices in this way, the following equations hold,

$$\begin{cases} d(0,1) = \max_{1 \leq i \leq m'} d(0,i), \\ d(k-1,k) = \min_{k \leq i \leq m'} d(k-1,i) \quad \text{for } k = 2, 3, \ldots, m'. \end{cases} \tag{3.5}$$

For each $k \in M'$ and $h = 1, 2, \ldots, c$, we define $y_k(h)$ by $y_k(h) = 1$ if there is a subset $J \subseteq \{1, 2, \ldots, k\}, 1 \in J$ with $q(J) = h$ and $y_k(p) = 0$ otherwise. We find routes $J_h$ for all $h = 1, 2, \ldots, c$ with $y_k(h) = 1$ by the following dynamic programming algorithm. Given two sequences $\sigma_1$ and $\sigma_2$ of items, we denote by $\sigma_1 \oplus \sigma_2$ the sequence that follows sequences $\sigma_1$ and $\sigma_2$ in this order.

**Procedure** ConstructRoute$_\varphi(M')$
**Input:** A set $M' = \{1, 2, \ldots, m'\}$ of items.
**Output:** Routes $J_h$ with $q(J_h) = h$ and costs $\varphi(J_h)$ for $h = 1, 2, \ldots, c$ with $y_{m'}(h) = 1$.

**1:** Renumber indices $k = 1, 2, \ldots, m'$ so that (3.5) holds;
**2:** $y_1(h) := 1$ and set route $J_h := [1]$ for $h = q(1)$; Compute $\varphi(J_h)$;
**3:** $y_1(h) := 0$ and $J_h := \emptyset$ for $j \in (\{1, 2, \ldots, c\} - \{q(1)\})$; Compute $\varphi(\emptyset)$;
**4:** **for** $k = 2, 3, \ldots, m'$ **do**
**5:** **for** $h = c, c-1, \ldots, 1$ **do**
**6:** $y_k(h) := y_{k-1}(h)$;
**7:** **if** $y_{k-1}(h) = 1$, $h + q(k) \leq c$, and $\varphi(J_h \oplus [k]) < \varphi(J_{h+q(k)})$ **then**
**8:** $y_k(h + q(k)) := 1$ and $J_{h+q(k)} := J_h \oplus [k]$

**9:** **end** /* if */
**10:** **end** /* for $p$ */
**11:** **end**. /* for $k$ */

Line 2 generates a route $J_h = [1]$ for $h = q(1)$. Lines 7-8 generate a new route $J_{h+q(k)}$ by adding item $k$ to route $J_h$ as its last item.

**Lemma 3.1.** *Every non empty route $J_h$ generated by* ConstructRoute$_\varphi$ *includes the first item* 1.

**proof:** Line 2 generates a route $J_h = [1]$. By Line 7, only when $y_{k-1}(h) = 1$ holds for $k \geq 2$ (i.e., there exists a route $J_h$ at $(k-1)$st iteration), route $J_h \oplus [k]$ which includes item $k$ can be generated. Hence, every non empty route includes the first item 1. □

The reason why we let routes include the first item 1, i.e., an item in the farthest customer, at each iteration is that leaving unprocessed items of distant customers would easily produce solutions of large travel cost.

### 3.3.3 Evaluation of routes

The main algorithm, DPRoute, simply outputs routes $\lambda_1, \ldots, \lambda_r$ chosen by procedure FindRoute$(M', g)$ from a set of routes generated by procedure ConstructRoute$_\varphi(M')$, where these procedures choose or generate routes with small function values $\varphi$. Therefore, choice of function $\varphi$ is very important to obtain a solution $\{\lambda_1, \ldots, \lambda_r\}$ with high quality. A simple setting of function $\varphi$ would be $\varphi(\lambda) = d(\lambda)$ for a route $\lambda$, which is the actual travel cost incurred by the route. However, as will be observed in some computational experiment in Section 3.4.3, solutions computed with this function $\varphi$ have a poor quality in both the number of vehicles and the total travel cost of routes. Our new idea is to estimate the minimum cost of routes to service the remaining items without computing any set of routes for those items. In general, it seems difficult to obtain a function that tells an optimal value of an instance without constructing any solutions. Fortunately, the DSDVRP seems to admit such a function. For a given set $M'$ of items and a route $\lambda$ from $M'$, we set function $\varphi$ to be

$$\varphi(\lambda) = d(\lambda) + E_1(\lambda) + E_2(\lambda), \tag{3.6}$$

where the first term $d(\lambda)$ is the travel cost required to service items in route $\lambda$, and the sum $E_1(\lambda) + E_2(\lambda)$ of the second and third terms estimates the minimum total travel cost of routes to service all the remaining items $M' - \lambda$.

We let $E_1(\lambda)$ (resp., $E_2(\lambda)$) represent an estimated total travel cost between the depot and customers (resp., among customers) required to service the remaining items in $M' - \lambda$.

We estimate the number of times to visit customer $j$ by $q(M_j \cap (M' - \lambda))/c$. The travel cost to visit customer $j$ is at least $2d(0, j)$. We then set $E_1(\lambda)$ to be

$$E_1(\lambda) = 2\sum_{j=1}^{n} d(0, j) \times \frac{q(M_j \cap (M' - \lambda))}{c}.$$

We observe that if $M_j \cap (M' - \lambda) \neq \emptyset$, i.e., at least one item in customer $j$ remains, then at least one vehicle must visit customer $j$, taking travel cost at least $2d(0, j)$ (even if $q(M_j \cap (M' - \lambda)) > 0$ is very small). Since it is difficult to know how vehicles visits the remaining customers in an optimal solution, we use the following simple formula for the third term $E_2(\lambda)$.

$$E_2(\lambda) = 0.15 \sum \{d(0, j) \mid j \text{ such that } M_j \cap (M' - \lambda) \neq \emptyset\},$$

where the coefficient 0.15 is determined through our computational tests. Note that the first $\nu - 1$ routes $\lambda_1, \ldots, \lambda_{\nu-1}$ have been computed and the incurred travel cost is $\sum_{i=1}^{\nu-1} d(\lambda_i)$ before computing the $\nu$ th route $\lambda_\nu$. Hence, for $\nu = 1, 2, \ldots, r$, the estimated minimum total travel cost $D_\nu$ is given by

$$\begin{cases} D_1 = \varphi(\lambda_1) & \text{for } \nu = 1, \\ D_\nu = \sum_{i=1}^{\nu-1} d(\lambda_i) + \varphi(\lambda_\nu) & \text{for } \nu = 2, 3, \ldots, r. \end{cases}$$

Note that $\varphi(\lambda_1)$ is an estimated minimum total travel cost of the given instance (i.e., $M' = M$). As will be seen in Section 3.4.3, the total travel cost of an output solution is well-estimated by $D_1, D_2, \ldots, D_{r-1}$, i.e., $D_\nu$ remains almost unchanged upon the completion of DPRoute.

### 3.3.4 Time and space complexities

In this section, we analyze the time and space complexities of our algorithms. We first show the following lemmas.

**Lemma 3.2.** ConstructRoute$_\varphi(M')$ *runs in* $O(cm' + n^2)$ *time and* $O(cm' + n^2)$ *space.*

**proof:** Renumbering indices $k = 1, 2, \ldots, m'$ in Line 1 can be done in $O(n^2 + m')$ time and $O(n^2 + m')$ space (it needs $O(n^2)$ space to save distances $d$). Since computing $\varphi(J_p \cup \{k\})$ in Line 7 can be done in $O(1)$ time by updating its value at $k - 1$, Lines 4-11 can be conducted in $O(cm')$ time and $O(cm' + n^2)$ space. Hence, ConstructRoute$_\varphi(M')$ runs in $O(cm' + n^2)$ time and $O(cm' + n^2)$ space. □

**Lemma 3.3.** FindRoute$_\varphi(M', g)$ *runs in* $O(cm' + n^2)$ *time and* $O(cm' + n^2)$ *space.*

**proof:** The time and space complexity of Line 1 was analyzed by Lemma 3.2. Line 2 of FindRoute$_\varphi(M', g)$ can be done in $O(c)$ time and $O(c)$ space. Hence, the time and space

complexities of FINDROUTE$_\varphi(M', g)$ are the same with those of CONSTRUCTROUTE$_\varphi(M')$. □

From the above lemmas, we have the following theorem.

**Theorem 3.1.** DPROUTE *runs in* $O((cm + n^2)r)$ *time and* $O(cm + n^2)$ *space.*

**proof:** Since the number of times to call FINDROUTE$_\varphi$ in DPROUTE is $r$, the total running time of DPROUTE is $O((cm+n^2)r)$. The computation space is $O(cm+n^2)$ as in FINDROUTE$_\varphi$. □

### 3.3.5 Reducing running time

In this section, we describe a way to reduce the running time to execute DPROUTE at the price of the solution quality. As shown in Section 3.3.4, the running time is affected by the size of capacity $c$, which is the maximum number routes constructed in CONSTRUCTROUTE$_\varphi$. We reduce the running time by diminishing the number of constructed routes.

In order to reduce the number of constructed routes, we introduce a parameter $w$ so that $\lceil c/w \rceil$ routes will be constructed in CONSTRUCTROUTE$_\varphi$. We redefine $J_{h'}$ for $h' = 1, 2, \ldots, \lceil c/w \rceil$ as a route which satisfies $(h'-1) \cdot w < q(J_{h'}) \leq h' \cdot w$, and redefine $y_k(h')$ for $h' = 1, 2, \ldots, \lceil c/w \rceil$ and $k = 1, 2, \ldots, m'$ as follows. For each $k \in M'$ and $h' = 1, 2, \ldots, \lceil c/w \rceil$, we define $y_k(h')$ by $y_k(h') = 1$ if there is a subset $J \subseteq \{1, 2, \ldots, k\}, 1 \in J$ with $(h' - 1) \cdot w < q(J) \leq h' \cdot w$, and $y_k(h') = 0$ otherwise. Procedures FINDROUTE$_\varphi(M',\ g)$ and CONSTRUCTROUTE$_\varphi(M')$ are modified as follows.

**Procedure** FINDROUTE$_\varphi(M',\ g,\ w)$
**Input:** A set $M'$ of items, parameters $g$ and $w$.
**Output:** A route $\lambda$.

**1:** Call CONSTRUCTROUTE$_\varphi(M', w)$ to construct routes $J_{h'}$ with $(h'-1) \cdot w < q(J_{h'}) \leq h' \cdot w$ and their costs $\varphi(J_{h'})$, $h' = 1, 2, \ldots, \lceil c/w \rceil$, where we let $J_{h'} = \emptyset$ if no such $J_{h'}$ exists. Let $h^*$ be the maximum $h'$ such that $J_{h'} \neq \emptyset$; (Note that such $h'$ necessarily exists since $|M'| > 0$ and for $k \in M'$, $1 \leq q(k) \leq c$ hold. )
**2:** Select a route $\lambda := \text{argmin}\{\varphi(J_{h'}) \mid \lfloor g \cdot c \rfloor \leq q(J_{h'}) \leq h^* \cdot w\}$ if $\lfloor g \cdot c \rfloor \leq h^* \cdot w$ and $\lambda := J_{h^*}$ otherwise.

**Procedure** CONSTRUCTROUTE$_\varphi(M', w)$
**Input:** A set $M'$ of items and a parameter $w$.
**Output:** Routes $J_{h'}$ with $(h'-1) \cdot w < q(J_{h'}) \leq h' \cdot w$ and costs $\varphi(J_{h'})$ for $h' = 1, 2, \ldots, \lceil c/w \rceil$ with $y_{m'}(h') = 1$.

**1:** Renumber indices $k = 1, 2, \ldots, m'$ so that (3.5) holds;

**2:** $y_1(h') := 1$ and set route $J_{h'} := [1]$ for $h' = \lceil q(1)/w \rceil$; Compute $\varphi(J_{h'})$;
**3:** $y_1(h') := 0$ and $J_{h'} := \emptyset$ for $h' \in \{1, 2, \ldots, \lceil c/w \rceil\} - \{\lceil q(1)/w \rceil\}$; Compute $\varphi(\emptyset)$;
**4:** **for** $k = 2, 3, \ldots, m'$ **do**
**5:** **for** $h' = \lceil c/w \rceil, \lceil c/w \rceil - 1, \ldots, 1$ **do**
**6:** $y_k(h') := y_{k-1}(h')$;
**7:** $u := \lceil (q(J_{h'}) + q(k))/w \rceil$;
**8:** **if** $y_{k-1}(h') = 1$, $u \leq \lceil c/w \rceil$ and $\varphi(J_{h'} \oplus [k]) < \varphi(J_u)$
**9:** $y_k(u) := 1$ and $J_u := J_{h'} \oplus [k]$
**10:** **end** /* if */
**11:** **end** /* for $h'$ */
**12:** **end**. /* for $k$ */

In Line 7, we introduced a variable $u$ that indicates an index $h'$ after adding item $k$ to $J_{h'}$. The time and space complexities satisfy the following theorem.

**Theorem 3.2.** DPROUTE *runs in* $O((cm/w+n^2)r)$ *time and* $O(cm/w+n^2)$ *space by using* $\text{FINDROUTE}_\varphi(M', g, w)$.

**proof:** Since the number of routes constructed in $\text{CONSTRUCTROUTE}_\varphi$ is reduced from $c$ to $\lceil c/w \rceil$, the time complexity of $\text{CONSTRUCTROUTE}_\varphi$ is reduced to $O(cm/w + n^2)$. Hence, DPROUTE runs in $O((cm/w + n^2)r)$ time. The computation space $O(cm + n^2)$ is reduced to $O(cm/w + n^2)$. □

## 3.4 Experimental results

This section reports experimental results on our algorithm and other heuristic algorithms that are explained in Chapter 2. The proposed algorithm was coded in C++ language and run on an IBM compatible PC (Intel Pentium 3.00GHz, 512MB RAM).

### 3.4.1 Performance of six algorithms

This section first shows experimental results on 20 instances. We generated the 20 instances so that feature of the instances are similar to instances provided from a huge manufacturer in Japan. Table 3.1 shows the feature of these instances, where $n$ is the number of customers, $m$ is the number of items, $q(M)/n$ is the average quantity per customer, and $q(M)/m$ is the average quantity per item. The row 'min' (resp., 'max') represents the minimum (resp., maximum) value for each column term. The vehicle capacity $c$ is set to be 250.

We compared DPROUTE with Dror and Trudeau's SDVRP algorithm and some classical construction heuristics for VRP. We also implemented a local search algorithm which improves

Table 3.1: Feature of instances

| | $n$ | $m$ | $q(M)/n$ | $q(M)/m$ |
|---|---|---|---|---|
| average | 55 | 832 | 367.6 | 23.3 |
| min | 15 | 455 | 286.8 | 12.8 |
| max | 77 | 1133 | 538.8 | 34.4 |

solutions obtained by these classical construction heuristics. The local search algorithm uses standard neighborhoods such as cross exchange neighborhood [86], 2-opt [52], and or-opt [66], which are introduced in Chapter 2. We set the maximum size of items exchanged by the cross exchange neighborhood to be 15.

Table 3.2 shows performance of DPRoute and other algorithms on these instances. In the table, NV represents the number of utilized vehicles, DIST is the total travel cost, and TIME is the execution time (second). Those are the average over the 20 instances. The first column SV is the well known saving algorithm [18], and SVLS is the saving algorithm followed by the local search algorithm. SW is the sweep algorithm [39] and SWLS is the sweep algorithm followed by the local search algorithm. We executed SV, SVLS, SW and SWLS algorithms by regarding each item as a customer in a VRP. DR is a representative SDVRP algorithm [28, 29]. In order to apply Dror and Trudeau's algorithm to the DSDVRP, we made a slight modification on it so that a customer demand is split according to a partition of the set of items in the customer. The last column DP is the proposed algorithm DPRoute with $g = 0.2$ and $w = 1$. Table 3.2 shows that DPRoute outperforms all of the other heuristics both in the number of vehicles utilized and the total travel cost. The computation time of DPRoute is larger than those of SV and SW. Table 3.3 shows a list of the total travel cost and computation time over 20 instances for the six algorithms. Column 'value' shows whether values on each row represent the total travel cost or computation time. Values of computation time are shown by second. DP outperforms the other algorithms on 18 instances. (Notation '*' indicates the best value for each instance. ) Section 3.4.4 will report results on computation time of DPRoute when parameter $w$ changes.

Table 3.2: Summary of performance of six algorithms

| | SV | SVLS | SW | SWLS | DR | DP |
|---|---|---|---|---|---|---|
| NV | 85.4 | 85.3 | 83.9 | 83.3 | 81.2 | 79.7 |
| DIST | 7546.2 | 7528.4 | 8277.8 | 7550.4 | 7334.6 | 7247.8 |
| TIME | 0.4 | 58.4 | 1.1 | 2.7 | 87.2 | 2.9 |

Table 3.3: Performance of six algorithms on the total travel cost and computation time

| instance | value | SV | SVLS | SW | SWLS | DR | DP |
|---|---|---|---|---|---|---|---|
| 1 | cost | 6094 | 6093 | 6618 | 6282 | 6069 | *6067 |
| | time | 0.66 | 111.53 | 1.88 | 3.97 | 55.42 | 3.92 |
| 2 | cost | 6026 | 5996 | 6744 | 5643 | *5401 | 5446 |
| | time | 0.41 | 86.20 | 1.19 | 3.27 | 157.97 | 2.95 |
| 3 | cost | 7440 | 7438 | 8466 | 7350 | 7314 | *7129 |
| | time | 0.36 | 39.95 | 1.02 | 3.17 | 93.25 | 2.95 |
| 4 | cost | 6959 | 6957 | 8213 | 7111 | 6922 | *6763 |
| | time | 0.44 | 78.34 | 1.33 | 3.36 | 37.56 | 4.22 |
| 5 | cost | 5958 | 5957 | 6495 | 6085 | 5825 | *5727 |
| | time | 0.41 | 31.49 | 1.20 | 3.95 | 521.63 | 3.95 |
| 6 | cost | 6316 | 6308 | 7073 | 6274 | 6254 | *6121 |
| | time | 0.59 | 101.97 | 1.78 | 3.92 | 56.69 | 4.74 |
| 7 | cost | 6789 | 6771 | 8249 | 6914 | 6690 | *6602 |
| | time | 0.50 | 66.38 | 1.47 | 4.23 | 126.52 | 4.74 |
| 8 | cost | 3535 | 3535 | 4164 | 3635 | 3469 | *3424 |
| | time | 0.22 | 29.48 | 0.69 | 1.55 | 7.38 | 1.83 |
| 9 | cost | 5079 | 5079 | 5349 | 5010 | 4965 | *4870 |
| | time | 0.33 | 37.53 | 0.97 | 2.50 | 68.41 | 2.77 |
| 10 | cost | 7440 | 7440 | 8421 | 7467 | 7092 | *6956 |
| | time | 0.27 | 25.70 | 0.78 | 2.69 | 95.56 | 2.61 |
| 11 | cost | 5107 | 5037 | 5784 | 5125 | 5022 | *4941 |
| | time | 0.36 | 68.72 | 1.09 | 2.34 | 19.58 | 3.23 |
| 12 | cost | 12051 | 12049 | 13457 | 12214 | 11750 | *11658 |
| | time | 0.36 | 36.88 | 1.08 | 3.80 | 308.34 | 3.11 |
| 13 | cost | 12187 | 12150 | 13444 | 12160 | 11909 | *11841 |
| | time | 0.66 | 133.05 | 1.94 | 5.03 | 44.89 | 4.91 |
| 14 | cost | 8537 | 8535 | 9377 | 8644 | 8349 | *8273 |
| | time | 0.30 | 49.52 | 0.91 | 2.25 | 51.95 | 2.44 |
| 15 | cost | 11502 | 11402 | 12198 | 11194 | 11009 | *10959 |
| | time | 0.50 | 113.31 | 1.45 | 3.36 | 70.81 | 3.81 |
| 16 | cost | 13862 | 13860 | 14409 | 13833 | 13357 | *13113 |
| | time | 0.22 | 29.06 | 0.69 | 1.80 | 25.73 | 2.17 |
| 17 | cost | 7019 | 7019 | 7664 | 7489 | 6933 | *6885 |
| | time | 0.28 | 41.80 | 0.89 | 1.25 | 0.95 | 1.47 |
| 18 | cost | 6110 | 6101 | 6312 | 5977 | 5957 | *5940 |
| | time | 0.09 | 15.42 | 0.30 | 0.50 | 0.08 | 0.55 |
| 19 | cost | 4807 | 4807 | 5129 | 4717 | *4580 | 4599 |
| | time | 0.16 | 23.33 | 0.48 | 0.70 | 0.14 | 0.84 |
| 20 | cost | 8106 | 8033 | 7988 | 7883 | 7825 | *7641 |
| | time | 0.19 | 49.38 | 0.55 | 0.94 | 0.47 | 1.00 |

### 3.4.2 Effect of parameter

Figure 3.1 shows the effect of parameter $g$ on the number of vehicles and the total travel cost. The horizontal axis represents the number of vehicles, while the vertical axis represents the total travel cost. The left and top side of the graph corresponds to the results of parameter $g = 1.0$ while the right and bottom side of the graph corresponds to the results of parameter $g = 10^{-6}$. From $g = 10^{-6}$ to 0.9, we incremented parameter $g$ by 0.1, and from $g = 0.9$ to 1.0, we incremented parameter $g$ by 0.01.

The curve in Fig. 3.1 indicates that, larger $g$, DPROUTE delivers solutions with a less number of vehicles at expense of travel cost.

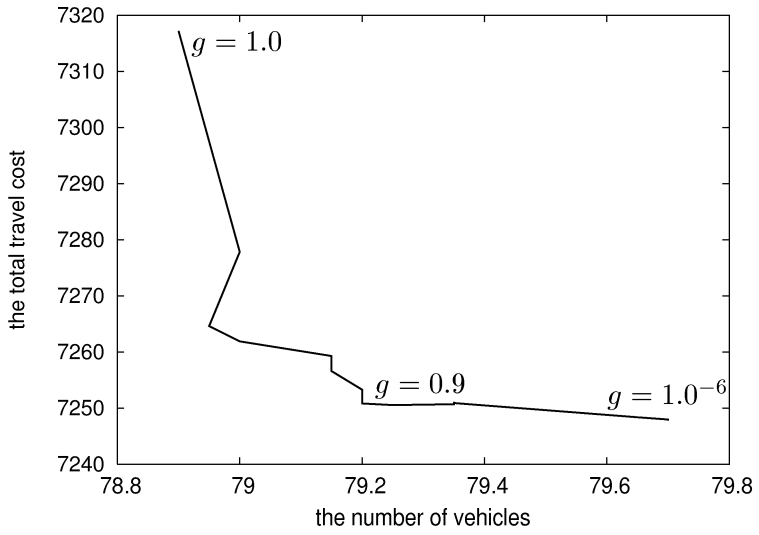



Figure 3.1: Effect of parameter $g$ on the number of vehicles and the travel cost.

### 3.4.3 Estimating travel cost

This section shows some experimental results which indicate the importance of choice of function $\varphi$ in algorithm DPROUTE.

Table 3.4 shows results on the performance of DPROUTE with function $\varphi(\lambda) = d(\lambda)$ (which is called naive DPROUTE) and DPROUTE with function $\varphi(\lambda) = d(\lambda) + E_1(\lambda) + E_2(\lambda)$ in (6). In DPROUTE, the total travel cost is the minimum when $g = 0.2$, while in the naive DPROUTE, the total travel cost is the minimum when $g = 0.6$. Table 3.4 shows that the total travel cost of the naive DPROUTE is considerably worse than that of DPROUTE by 21%.

Figure 3.2 shows experimental result by DPROUTE for an instance in the 20 instances, where the horizontal axis represents $\nu$ of DPROUTE and the vertical axis represents the total travel cost. The dashed curve shows the total travel cost $\sum_{1 \leq i \leq \nu} d(\lambda_i)$ of routes $\lambda_1, \ldots, \lambda_\nu$

Table 3.4: Effect of function $\varphi$ on the performance of DPROUTE

| | DP($\varphi = d$) | | DP($\varphi = d + E_1 + E_2$) | |
|---|---|---|---|---|
| | $g = 0.6$ | $g = 1.0$ | $g = 0.2$ | $g = 1.0$ |
| NV | 85.8 | 78.9 | 79.7 | 78.9 |
| DIST | 8811.4 | 9532.7 | 7247.8 | 7317.2 |

constructed after the $\nu$th iteration in DPROUTE, while the solid curve shows cost $D_\nu$, i.e., the sum of the total travel cost $\sum_{1 \le i \le \nu} d(\lambda_i)$ of constructed routes and the estimated travel cost $E_1(\lambda_\nu) + E_2(\lambda_\nu)$ to service the remaining items in $M'$. It should be noted that the solid curve stays horizontal. This means that the total travel cost of a final solution by DPROUTE is always well estimated by function $\varphi$ in (6) during the execution of DPROUTE. In fact, the total travel cost at $\nu = 0$ (i.e., the initial estimated travel cost) is 13059 and that of the final (i.e., the actual travel cost of the solution) is 13113, where the difference is only 0.42%. For other instances, the differences are very small, which is only 3.2% on average.

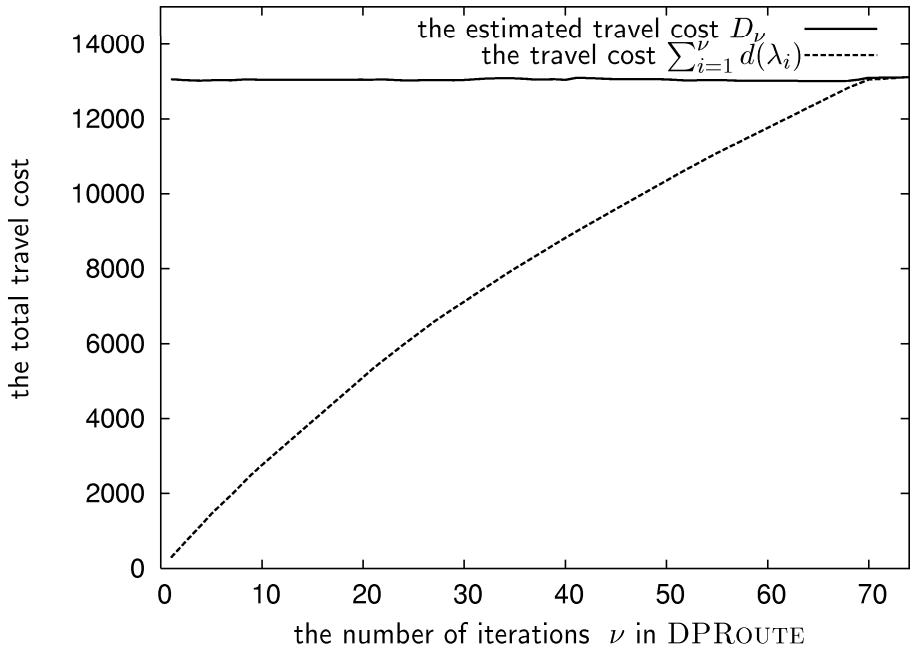



Figure 3.2: Transition of the estimated travel cost $D_\nu$ and travel cost $\sum_{i=1}^{\nu} d(\lambda_i)$ in DPROUTE.

### 3.4.4 Effect of reducing the size of columns

The curve in Fig. 3.3 (resp., Fig. 3.4) shows a trade-off between the execution time and the total travel cost of routes (resp., the number of vehicles) when parameter $w$ changes.

The total travel cost and the number of vehicles are the averages over the 20 instances. The left and top side of the curve corresponds to the results of $w = 18$, while the right and bottom side of the graph corresponds to the results of $w = 1$. The curves show that, the larger $w$ is, the faster DPROUTE runs at the price of solution quality. When $w = 12$, the average of total travel cost is 7309, the number of vehicles is 80.4, and the computation time is 0.38 second. The travel cost and the number of vehicles are the smallest, and the execution time is the shortest among all other heuristics.

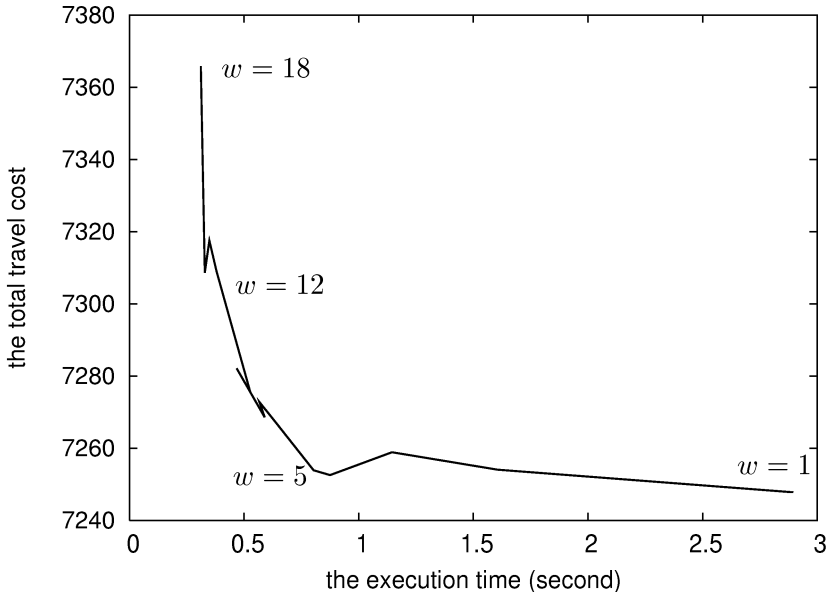



Figure 3.3: Effect of parameter $w$ on the execution time and the total travel cost.

### 3.4.5 Results on artificial instances

In this section, we show experimental results on artificial instances. In order to investigate influence of the number $n$ of customers and quantity per a customer on the total travel cost, instances are generated for each $n = 40, 60, 80, 100$ and $\sum_{k \in M} q(k)/n = 150, 200, 250, 300, 350, 400$. Each customer demand is randomly chosen from interval [0.1 $\sum q(k)/n$, 1.9 $\sum q(k)/n$]. The number of items for a customer is randomly chosen from interval [5, 15]. By using the number of items for a customer $i$, the average quantity of an item is determined for customer $i$, and we denote the number by $q_i$. The quantity of an item is randomly chosen from interval [0.5 $q_i$, 1.5 $q_i$]. For each $n = 40, 60, 80, 100$ and

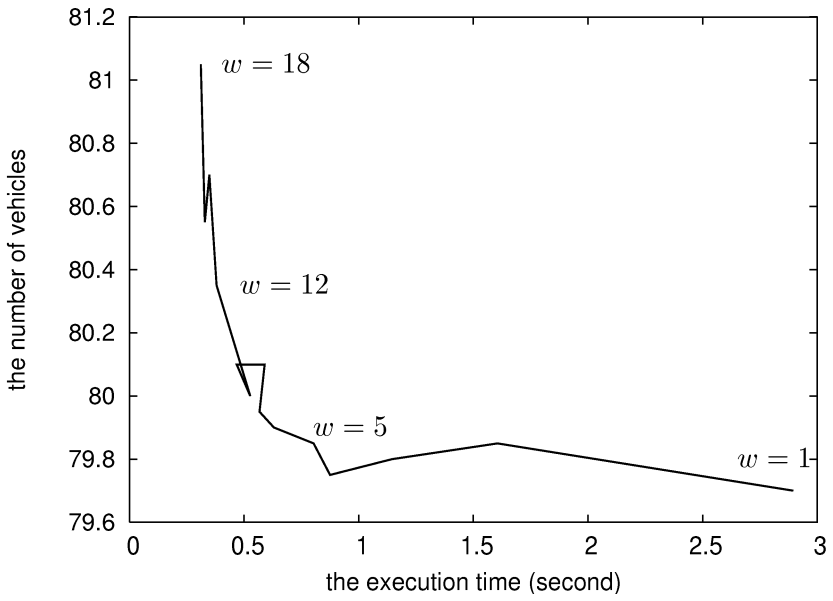



Figure 3.4: Effect of parameter $w$ on the execution time and the number of vehicles.

$\sum_{k \in M} q(k)/n = 150, 200, 250, 300, 350, 400,$ we generated 10 instances. In Table. 3.5 and Table. 3.6, column DP represents the average of the total cost or computation time on algorithm DPRoute over 10 instances, and SVLS and SWLS represent the average of the total cost or computation time on SVLS and SWLS over 10 instances, respectively. From these results, we observe that DPRoute produces better solutions for instances where the number of customers is large, and quantity for a customer is large than SVLS and SWLS.

Table 3.5: Performance of three algorithms on instances where the number of customers is 40 or 60

| instance | $n$ | $\sum q(k)/n$ | value | DP | SVLS | SWLS |
|---|---|---|---|---|---|---|
| 1 | 40 | 150 | cost | 255 | 250 (98%) | 260 (102%) |
| | | | time | 0.64 | 17.84 | 0.50 |
| 2 | 40 | 200 | cost | 325 | 327 (101%) | 337 (104%) |
| | | | time | 0.76 | 11.74 | 0.58 |
| 3 | 40 | 250 | cost | 393 | 392 (100%) | 411 (104%) |
| | | | time | 0.88 | 10.03 | 0.72 |
| 4 | 40 | 300 | cost | 468 | 467 (100%) | 478 (102%) |
| | | | time | 0.95 | 8.98 | 0.87 |
| 5 | 40 | 350 | cost | 540 | 547 (101%) | 551 (102%) |
| | | | time | 1.01 | 7.35 | 0.99 |
| 6 | 40 | 400 | cost | 616 | 615 (100%) | 622 (101%) |
| | | | time | 1.04 | 6.22 | 1.20 |
| 7 | 60 | 150 | cost | 358 | 360 (101%) | 374 (104%) |
| | | | time | 1.50 | 45.88 | 1.19 |
| 8 | 60 | 200 | cost | 464 | 465 (100%) | 482 (104%) |
| | | | time | 1.79 | 30.46 | 1.42 |
| 9 | 60 | 250 | cost | 566 | 574 (101%) | 589 (104%) |
| | | | time | 2.05 | 23.50 | 1.68 |
| 10 | 60 | 300 | cost | 672 | 679 (101%) | 694 (103%) |
| | | | time | 2.28 | 22.62 | 2.10 |
| 11 | 60 | 350 | cost | 780 | 792 (102%) | 800 (103%) |
| | | | time | 2.41 | 18.01 | 2.35 |
| 12 | 60 | 400 | cost | 888 | 906 (102%) | 915 (103%) |
| | | | time | 2.58 | 16.53 | 2.83 |

Table 3.6: Performance of three algorithms on instances where the number of customers is 80 or 100

| instance | $n$ | $\sum q(k)/n$ | value | DP | SVLS | SWLS |
|---|---|---|---|---|---|---|
| 13 | 80 | 150 | cost | 472 | 474 (101%) | 487 (103%) |
| | | | time | 2.85 | 93.75 | 2.17 |
| 14 | 80 | 200 | cost | 613 | 619 (101%) | 639 (104%) |
| | | | time | 3.38 | 65.29 | 2.60 |
| 15 | 80 | 250 | cost | 749 | 768 (102%) | 787 (105%) |
| | | | time | 3.83 | 44.81 | 3.04 |
| 16 | 80 | 300 | cost | 894 | 912 (102%) | 930 (104%) |
| | | | time | 4.27 | 46.72 | 3.67 |
| 17 | 80 | 350 | cost | 1039 | 1062 (102%) | 1076 (104%) |
| | | | time | 4.63 | 34.20 | 4.35 |
| 18 | 80 | 400 | cost | 1188 | 1209 (102%) | 1225 (103%) |
| | | | time | 4.84 | 33.33 | 5.27 |
| 19 | 100 | 150 | cost | 588 | 596 (101%) | 615 (105%) |
| | | | time | 4.65 | 155.62 | 3.40 |
| 20 | 100 | 200 | cost | 769 | 781 (102%) | 801 (104%) |
| | | | time | 5.55 | 99.54 | 4.18 |
| 21 | 100 | 250 | cost | 945 | 965 (102%) | 989 (105%) |
| | | | time | 6.23 | 77.39 | 4.92 |
| 22 | 100 | 300 | cost | 1126 | 1151 (102%) | 1171 (104%) |
| | | | time | 6.88 | 73.99 | 6.42 |
| 23 | 100 | 350 | cost | 1312 | 1347 (103%) | 1362 (104%) |
| | | | time | 7.49 | 63.18 | 7.38 |
| 24 | 100 | 400 | cost | 1495 | 1541 (103%) | 1549 (104%) |
| | | | time | 7.93 | 51.21 | 8.93 |

# Chapter 4

# Worst-Case Analysis on the Optimal Cost between PDP with Transfer and PDP

## 4.1 Introduction

In this chapter, we discuss the pickup and delivery problem with transfer (PDPT). PDPT introduces a set of transshipment points to PDP. At transshipment points, each vehicle is allowed to temporarily drop some of the loads on it, and the loads will be picked it up later by the vehicle or another vehicle. This chapter analyzes the maximum travel cost that can be saved by introducing a transshipment point to PDP. We show that the bounds are in proportion to square root of the number of cycles in an optimal PDPT solution and also square root of the number of requests. The bounds are also hold for the pickup and delivery problem with consecutive pickups and deliveries (PDPC) to the pickup and delivery problem with transfer with consecutive pickups and deliveries (PDPTC). We furthermore present an instance where the bound is tight for PDPC to PDPTC.

## 4.2 Problem definition

This section defines PDP and PDPT that are introduced in Chapter 1 again, and gives additional definitions. Depots, customers and transshipment points to be visited by vehicles are represented by vertices in an edge-weighted complete digraph with a vertex set $V$, which is given by distance functions $d(u, v)$ for all ordered pairs of vertices $u$ and $v$. Distance $d(u, v)$ is a nonnegative real number, and in general asymmetric, i.e., $d(u, v) \neq d(v, u)$ may hold. Distance $\{d(u, v) \mid u, v \in V\}$ may not satisfy the Triangle Inequality. Let $Q$ denote a set of depots, $R$ denote a set of of requests. Each request $r = \{r^+, r^-\} \in R$ consists of a pickup

action $r^+$, a *delivery action* $r^-$, and a quantity $q(r)$ of loads for the request. We may denote the vertex where request $r$ is picked up by $r^+$ (resp., delivered by $r^-$), and call the vertex $r^+$ a *pickup point* (resp., $r^-$ a *delivery point*). In this chapter, we call pickup and delivery points *customers*. Let $p$ be the number of requests in $R$.

Every vehicle has capacity $c$, where $c$ is a nonnegative real number, and each vehicle must start from its predetermined depot, and return to the depot after serving requests assigned to the vehicle. We assume that any number of vehicles is available at each depot.

Given $v_0, v_1, v_2, \ldots, v_u \in V$, where $v_0, v_u \in Q$, *route* $\sigma$ is a sequence of vertices in $V$, and its travel cost $d(\sigma)$ is defined to be

$$d(\sigma) = \sum_{0 \leq i \leq u-1} d(v_i, v_{i+1}).$$

A *solution* $s$ is a set of routes that serves all requests in $R$, and its cost $cost(s)$ is defined by the sum of the travel costs of the routes $\sigma$ in $s$.

We review the pickup and delivery problem below.

**Pickup and Delivery Problem (PDP):**
**Input:** An instance $I = (Q, R, c)$.
**Output:** A minimum cost solution that satisfies the following constraints (see Fig. 1.1):

(a) the total quantity of loads during a route does not exceed vehicle capacity $c$ at any time;

(b) each request $r$ is served by exactly one vehicle, and the actions $r^+$ and $r^-$ are taken only once by the vehicle;

(c) (*coupling constraint*) actions $r^+$ and $r^-$ of each request $r$ must appear in the same route, and no request $r$ is allowed to be temporarily dropped at any vertices; and

(d) (*precedence constraint*) for each request $r$, action $r^+$ must be taken before action $r^-$.

The pickup and delivery problem with transfer (PDPT) introduces a set of transshipment points to PDP. In this thesis, we assumes that the number of transshipment points is one in order to make it possible to analyze the ratio of travel cost of PDPT to that of PDP. Let $t$ denote the transshipment point. At the transshipment point $t$, each vehicle is allowed to temporarily drop some of the loads on it and the loads will be picked it up later by the vehicle or some other vehicle [56, 62, 78]. This indicates that PDPT is obtained from PDP by dropping constraint (c).

**PDP with Transfer (PDPT):**
**Input:** An instance $I = (Q, R, c, t)$.

**Output:** A minimum cost solution that satisfies constraints (a), (b) and (d).

Each vehicle can visit transshipment point $t$ more than once. Suppose we are given a sequence of vertices $v_0, v_1, v_2, \ldots, v_u \in V$. If it hold $v_0 = v_u$ and $v_i \neq v_j$ for $i \neq 0$ or $j \neq u$, i.e., the sequence of vertices has only vertex $v_0 = v_u$ in common, then we call the sequence a *cycle*. If a route $\sigma$ visits the transshipment point $k$ times, then route $\sigma$ is supposed to include $k$ cycles, where only a cycle includes a depot and all cycles have transshipment point $t$ in common. Note that if each vehicle can visit transshipment point $t$ at most once, then all cycles correspond to routes.

If cycle $\sigma$ includes a depot $q \in Q$, then let $\sigma^+$ stand for the subpath of cycle $\sigma$ from the vertex succeeding $q$ to the vertex preceding $t$ on cycle $\sigma$, and $\sigma^-$ stand for the subpath from the vertex succeeding $t$ to the vertex preceding $q$ on cycle $\sigma$. Given sequences $\sigma_1, \sigma_2, \ldots, \sigma_s$ of vertices, we denote by $\sigma_1 \oplus \sigma_2 \oplus \cdots \oplus \sigma_s$ the sequence of vertices that follows $\sigma_1, \sigma_2, \ldots, \sigma_s$ in this order. If cycle $\sigma$ includes a depot $q \in Q$, then cycle $\sigma$ is expressed by $\sigma = q \oplus \sigma^+ \oplus t \oplus \sigma^- \oplus q$. If cycle $\sigma$ does not include a depot, then we let $\sigma^+ = \sigma - \{t\}$ and $\sigma^- = \emptyset$. Fig. 4.1 illustrates an example of a PDPT solution with two routes (three cycles), where $q_1$ and $q_2$ are depots. The first route is $q_1 \oplus \sigma_1^+ \oplus t \oplus \sigma_1^- \oplus q_1$, and the second route is $q_2 \oplus \sigma_2^+ \oplus t \oplus \sigma_3 \oplus t \oplus \sigma_2^- \oplus q_2$.

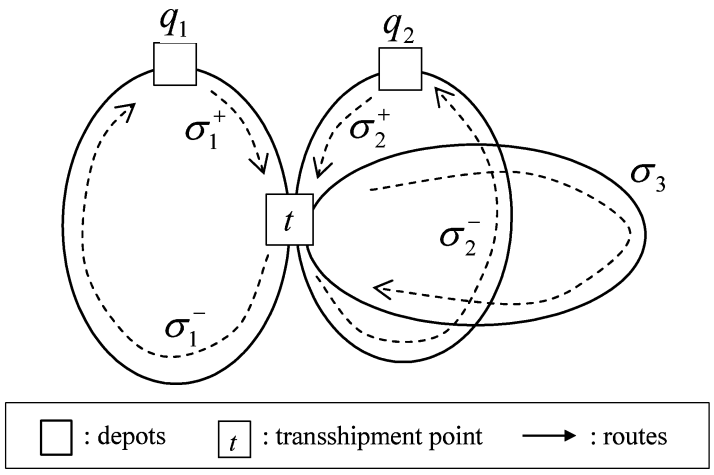



Figure 4.1: An example of two PDPT routes.

## 4.3 First-Fit Procedure

In this section, we introduce a well-known First-Fit procedure that is used in conversion algorithms in Section 4.4.2 and 4.4.3. We are given $n$ bins with capacity $c$, and a set $I$ of items. Item $i \in I$ has quantity $q(i)$. Procedure FIRSTFIT inserts each item into one of the bins so that each item is not split and the total quantity of items for each bin does not exceed

$c$. We denote by $I_k$ a set of items that are inserted into $k$-th bin for $k = 1, \ldots, n$.

**Procedure** FIRSTFIT$(I, c, n)$
**Input:** A set $I$ of items, capacity $c$ and $n$ bins, where item $i \in I$ has quantity $q(i)$.
**Output:** Sets $I_k$, $k = 1, \ldots, n'(\leq n)$ of items such that $I_1 \cup I_2 \cup \cdots \cup I_{n'} = I$ and $I_k \neq \emptyset$ for $k = 1, \ldots, n'$.

**1:** $I_k := \emptyset$ for $k = 1, \ldots, n$;
**2:** **for** $i = 1, \ldots, |I|$ **do**
**3:** Search $I_1, \ldots, I_n$ and select $I_k \in \{I_1, \ldots, I_n\}$ such that $q(I_k \cup \{i\}) \leq c$ and $q(I_j \cup \{i\}) > c$ for $j = 1, \ldots, k-1$;
**4:** $I_k := I_k \cup \{i\}$
**5:** **end**. /* for */

For the First-Fit procedure, the next theorem is known.

**Theorem 4.1.** [45] *Given $n$ bins with capacity $c$ and a set $I$ of items where item $i \in I$ has quantity $q(i)$. Let $n'$ be the number of bins that at least one of the items are inserted in* FIRSTFIT. *Then it holds*

$$n' \leq \frac{2}{c} \sum_{i=1}^{|I|} q(i).$$

**Proof:** We show the theorem by contradiction. Suppose that $n' \cdot c/2 > \sum_{i=1}^{|I|} q(i)$. Then there exists a bin $X$ such that the total quantity of items in $X$ is less than $c/2$. If there exists a bin $Y$ other than bin $X$ such that the total quantity of items in bin $Y$ is less than $c/2$, then the items in bins $X$ and $Y$ must be inserted in the same bin, which is a contradiction. If $X$ is the only bin whose total quantity is less than $c/2$, then there exists a bin $Z$ such that the sum of quantity of bins $X$ and $Z$ is less than or equal to capacity $c$, which is also a contradiction. □

## 4.4 Upper bound on travel cost of PDP solutions to PDPT solutions

Given an optimal PDPT solution $s$, let $m = indeg(t)$, i.e., the number of cycles in $s$. Given an instance $I$ and problem PRB, let $opt_{\text{PRB}}(I)$ denote the optimal cost to PRB with instance $I$. This section proves the following two theorems.

**Theorem 4.2.** *Suppose that each vehicle can visit the transshipment point at most once. Let $I = (Q, R, c, t)$ be an instance. Let $m$ denote the number of cycles in an optimal PDPT*

*solution to $I$. Then it holds*

$$opt_{\mathrm{PDP}}(I) < (6\lceil\sqrt{m}\rceil + 1) \cdot opt_{\mathrm{PDPT}}(I).$$

□

**Theorem 4.3.** *Suppose that each vehicle can visit the transshipment point any number of times. Let $I = (Q, R, c, t)$ be an instance. Let $m$ denote the number of cycles in an optimal PDPT solution to $I$. Then it holds*

$$opt_{\mathrm{PDP}}(I) < (6\lceil\sqrt{m}\rceil + 2) \cdot opt_{\mathrm{PDPT}}(I).$$

□

In order to prove these theorems, we convert an optimal PDPT solution $s$ to a PDP solution in which no vehicles visit transshipment point $t$, and we show that the travel cost for the constructed solution is less than $(6\lceil\sqrt{m}\rceil + 1) \cdot opt_{\mathrm{PDPT}}(I)$ if each vehicle can visit the transshipment point at most once, and less than $(6\lceil\sqrt{m}\rceil + 2) \cdot opt_{\mathrm{PDPT}}(I)$ if each vehicle can visit the transshipment point any number of times. For simplicity, we assume that every route visits transshipment point $t$ since if there exists a route that does not visit transshipment point $t$, the route need not to be converted, which does not lead to increase of travel cost.

### 4.4.1 Division of cycles in a PDPT solution

This section gives some definitions and a lemma that is used to analyze the maximum travel cost that can be saved by using a transshipment point to PDP in Section 4.4.2 and 4.4.3.

Let $\pi$ be the set of cycles in a PDPT solution. Given two cycles $\sigma_i$ and $\sigma_j$ in $\pi$, let $R(\sigma_i, \sigma_j)$ stand for the set of requests that are picked up at customers on cycle $\sigma_i$ and delivered to customers on cycle $\sigma_j$, and let $R(\sigma_i, \pi)$ stand for the set of requests that are picked up at customers on cycle $\sigma_i$ and delivered to customers on all cycles in $\pi$. Let $q(\sigma_i, \sigma_j) = \sum_{r \in R(\sigma_i, \sigma_j)} q(r)$ for $i, j = 1, \ldots, m$. For cycle $\sigma \in \pi$, since all loads of requests in $R(\sigma, \pi) - \{(R(\sigma^+, \sigma^+) \cup R(\sigma^-, \sigma^-)\}$ are on a vehicle when a vehicle reaches $t$ on $\sigma$ in a PDPT solution, we have

$$q(\sigma, \pi) - \{q(\sigma^+, \sigma^+) + q(\sigma^-, \sigma^-)\} \leq c. \tag{4.1}$$

We divide set $\pi$ of cycles into $\lceil\sqrt{m}\rceil$ subsets $\pi_i$, $i = 1, \ldots, \lceil\sqrt{m}\rceil$, so that each subset $\pi_i$ includes at most $\lceil\sqrt{m}\rceil$ cycles. Let $\pi_i = \{\sigma_{i,1}, \sigma_{i,2}, \ldots, \sigma_{i,\lceil\sqrt{m}\rceil}\}$ for $i = 1, \ldots, \lceil\sqrt{m}\rceil$. If $\lceil\sqrt{m}\rceil \cdot \lceil\sqrt{m}\rceil > m$, then we assume that $\sigma_{i,j} = \emptyset$ for some $i, j \in [1, \lceil\sqrt{m}\rceil]$. If cycle $\sigma_{i,j}$ includes a depot, then we denote the depot by $p_{i,j}$. Fig. 4.2 illustrates an example of routes in a PDPT solution with $m = 9$.

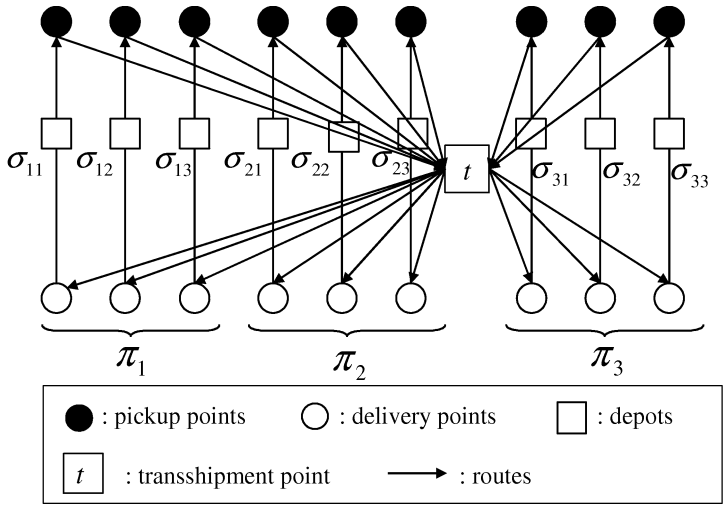



Figure 4.2: An example of PDPT routes with $m = 9$.

Let $R(\pi_i, \pi_j)$ stand for the set of requests that are picked up at customers on $\pi_i$ and delivered to customers on $\pi_j$. Let $q(\pi_i, \pi_j) = \sum_{r \in R(\pi_i, \pi_j)} q(r)$ and $d(\pi_i) = \sum_{\sigma \in \pi_i} d(\sigma)$. If $\pi$ be the set of cycles in an optimal PDPT solution $s$, then it is trivial to see that it holds

$$opt_{\mathrm{PDPT}}(I) = \sum_{i=1}^{\lceil \sqrt{m} \rceil} d(\pi_i). \tag{4.2}$$

We introduce the following Lemma.

**Lemma 4.1.** *Let $m$ be the number of cycles in a PDPT solution. Given sets $\pi_i$ of routes for $i = 1, \ldots, \lceil \sqrt{m} \rceil$, it holds*

$$\begin{aligned} q(\pi_i, \pi) &- \sum_{\sigma \in \pi_i} \{q(\sigma^+, \sigma^+) + q(\sigma^-, \sigma^-)\} \leq \lceil \sqrt{m} \rceil \cdot c \\ \textit{for } i &= 1, \ldots, \lceil \sqrt{m} \rceil. \end{aligned} \tag{4.3}$$

**Proof:** Inequality (4.1) and the assumption that $|\pi_i| \leq \lceil \sqrt{m} \rceil$ ensure the lemma. □

### 4.4.2 Case 1: visit a transshipment point at most once

In order to prove Theorem 4.2, we first consider the case that each vehicle can visit transshipment point $t$ at most once. The algorithm to convert a PDPT solution to a PDP solution is described as follows. In a PDP solution, route $\sigma'_{i,j}$ first follows cycles (routes) in $\pi_i$, and then follows cycles (routes) in $\pi_j$ to pick up and deliver requests in $R(\pi_i, \pi_j)$ if $i \neq j$ and

$R(\pi_i, \pi_j) - \bigcup_{\sigma\in\pi}\{R(\sigma^+, \sigma^+) \cup R(\sigma^-, \sigma^-)\}$ if $i = j$, for $i, j = 1, \ldots, \lceil\sqrt{m}\rceil$. Another route $\sigma'$ follows all cycles (routes) in $\pi$ in order to serve requests in $\bigcup_{\sigma\in\pi}\{R(\sigma^+, \sigma^+) \cup R(\sigma^-, \sigma^-)\}$.

**Algorithm** CONVERT
**Input:** An instance $I$ with a set $R$ of requests and a set $\pi$ of PDPT routes, each of which visits transshipment point $t$ at most once.
**Output:** A set $\{\sigma'_{i,j} \mid i, j = 1, 2, \ldots, \lceil\sqrt{m}\rceil\} \cup \{\sigma'\}$ of PDP routes.

**1:** **for** $i = 1, 2, \ldots, \lceil\sqrt{m}\rceil$ **do**
**2:** **for** $j = 1, 2, \ldots, \lceil\sqrt{m}\rceil$ **do**
**3:** **if** $i \neq j$ **then**
**4:** $\{R_1, \ldots, R_{n'}\} := FirstFit(R(\pi_i, \pi_j), c, m)$
**5:** **else**
**6:** $\{R_1, \ldots, R_{n'}\} := FirstFit(R(\pi_i, \pi_i) - \bigcup_{\sigma\in\pi_i}\{R(\sigma^+, \sigma^+) \cup R(\sigma^-, \sigma^-)\}, c, m)$
**7:** **end**; /* if */
**8:** **for** $k = 1, 2, \ldots, n'$ **do**
**9:** Route $\sigma'_{i,j}$ follows routes in $\pi_i$ to pick up requests in $R_k$;
**10:** Route $\sigma'_{i,j}$ follows routes in $\pi_j$ to deliver requests in $R_k$
**11:** **end** /* for */
**12:** **end** /* for */
**13:** **end**; /* for */
**14:** Route $\sigma'$ follows all routes in $\pi$ in order to serve requests in $\bigcup_{\sigma\in\pi}\{R(\sigma^+, \sigma^+) \cup R(\sigma^-, \sigma^-)\}$.

In Line 9, route $\sigma'_{i,j}$ is represented by $\sigma'_{i,j} := \sigma'_{i,j} \cup p_{i,1} \oplus \sigma^+_{i,1} \oplus t \oplus \sigma^-_{i,2} \oplus p_{i,2} \oplus \sigma^+_{i,2} \oplus t \oplus \cdots \oplus t \oplus \sigma^-_{i,\lceil\sqrt{m}\rceil} \oplus p_{i,\lceil\sqrt{m}\rceil} \oplus \sigma^+_{i,\lceil\sqrt{m}\rceil} \oplus t \oplus \sigma^-_{i,1} \oplus p_{i,1}$. In Line 10, route $\sigma'_{i,j}$ follows routes in $\pi_j$ in the same way. In Line 14, route $\sigma'$ is represented by $\sigma' := p_{1,1} \oplus \sigma^+_{1,1} \oplus t \oplus \sigma^-_{1,2} \oplus p_{1,2} \oplus \sigma^+_{1,2} \oplus t \oplus \cdots \oplus t \oplus \sigma^-_{\lceil\sqrt{m}\rceil,\lceil\sqrt{m}\rceil} \oplus p_{\lceil\sqrt{m}\rceil,\lceil\sqrt{m}\rceil} \oplus \sigma^+_{\lceil\sqrt{m}\rceil,\lceil\sqrt{m}\rceil} \oplus t \oplus \sigma^-_{1,1} \oplus p_{1,1}$. We show the following lemma.

**Lemma 4.2.** *PDP routes obtained by* CONVERT *serves all requests in* $R$.

**Proof:** By iterating Line 3-11 for $i, j = 1, \ldots, \lceil\sqrt{m}\rceil$, all requests in $R - \bigcup_{\sigma\in\pi}\{R(\sigma^+, \sigma^+) \cup R(\sigma^-, \sigma^-)\}$ are served. All requests in $\bigcup_{\sigma\in\pi}\{R(\sigma^+, \sigma^+) \cup R(\sigma^-, \sigma^-)\}$ are served in Line 14. Thus we have the lemma. □

We now analyze travel cost of routes that are constructed by CONVERT. The following lemma is used to prove Theorem 4.2.

**Lemma 4.3.** *For given* $i \in [1, \lceil\sqrt{m}\rceil]$, *let* $d'_i$ *be the travel cost that is required to follow routes in* $\pi_i$ *in Line 9 of* CONVERT *for all* $j = 1, \ldots, \lceil\sqrt{m}\rceil$. *Then it holds*

$$d'_i < 3\lceil\sqrt{m}\rceil \cdot d(\pi_i).$$

**Proof:** For given two integers $i, j \in [1, \lceil\sqrt{m}\rceil]$, Theorem 4.1 gives $n' \leq \lceil 2q(\pi_i, \pi_j)/c\rceil$ for $i \neq j$, and $n' \leq \lceil 2(q(\pi_i, \pi_j) - \sum_{\sigma\in\pi_i}\{q(\sigma^+, \sigma^+) + q(\sigma^-, \sigma^-)\})/c\rceil$ for $i = j$.
Let $q'(\pi_i, \pi_j) = q(\pi_i, \pi_j) - \sum_{\sigma\in\pi_i}\{q(\sigma^+, \sigma^+) + q(\sigma^-, \sigma^-)\}$. Then we obtain

$$\begin{aligned} d'_i &\leq \sum_{j=1}^{\lceil\sqrt{m}\rceil} \lceil 2q'(\pi_i, \pi_j)/c\rceil \cdot d(\pi_i) \\ &< \sum_{j=1}^{\lceil\sqrt{m}\rceil} (2q'(\pi_i, \pi_j)/c + 1) \cdot d(\pi_i) \\ &= (\sum_{j=1}^{\lceil\sqrt{m}\rceil} 2q'(\pi_i, \pi_j)/c + \lceil\sqrt{m}\rceil) \cdot d(\pi_i). \end{aligned}$$

By applying (4.3), we have

$$\begin{aligned} d' &< (2\lceil\sqrt{m}\rceil + \lceil\sqrt{m}\rceil) \cdot d(\pi_i) \\ &= 3\lceil\sqrt{m}\rceil \cdot d(\pi_i). \end{aligned}$$

□

We are now ready to prove Theorem 4.2.

**Proof of Theorem 4.2:** For given $j \in [1, \lceil\sqrt{m}\rceil]$, let $d''_j$ be the travel cost that is required to follow routes in $\pi_j$ in Line 10 of CONVERT for all $i = 1, \ldots, \lceil\sqrt{m}\rceil$. Lemma 4.3 is easily extended to show that

$$d''_j < 3\lceil\sqrt{m}\rceil \cdot d(\pi_j).$$

Travel cost of route $\sigma'$ that is constructed in Line 14 is equal to $d(\pi)$. Thus we obtain

$$\begin{aligned} opt_{\mathrm{PDP}}(I) &\leq \sum_{i=1}^{\lceil\sqrt{m}\rceil} d'_i + \sum_{j=1}^{\lceil\sqrt{m}\rceil} d''_j + d(\pi) \\ &< 6\lceil\sqrt{m}\rceil \cdot \sum_{i=1}^{\lceil\sqrt{m}\rceil} d(\pi_i) + d(\pi). \end{aligned}$$

By applying (4.2), it holds

$$opt_{\mathrm{PDP}}(I) < (6\lceil\sqrt{m}\rceil + 1) \cdot opt_{\mathrm{PDPT}}(I).$$

□

If we use the number $p$ of requests, the next theorem holds by using $m \leq p$.

**Theorem 4.4.** *Suppose that each vehicle can visit the transshipment point at most once. Let $I = (Q, R, c, t)$ be an instance with $p$ requests. Then it holds*

$$opt_{\mathrm{PDP}}(I) < (6\lceil\sqrt{p}\rceil + 1) \cdot opt_{\mathrm{PDPT}}(I).$$

□

### 4.4.3 Case 2: visit a transshipment point any number of times

We next consider the case that each vehicle can visit transshipment point $t$ any number of times, and prove Theorem 4.3. The difference between Case 1 and Case 2 is that some cycles may not include a depot in Case 2, while all cycles include a depot in Case 1. We describe an algorithm to convert a PDPT solution, where each vehicle can visit transshipment point $t$ any number of times, to a PDP solution below. All requests are served by one vehicle, which moves from a depot to transshipment point $t$, serves all requests, and finally returns to the depot.

**Algorithm** CONVERT2
**Input:** An instance $I$ with a set $R$ of requests and a set $\pi$ of cycles in a PDPT solution.
**Output:** A PDP route $\sigma'$.

**1:** Route $\sigma'$ starts from depot $q \in \sigma$, where $\sigma \in \pi$ is a cycle that includes a depot, follows path $\sigma^+$, and reaches transshipment point $t$;
**2:** **for** $i = 1, 2, \ldots, \lceil\sqrt{m}\rceil$ **do**
**3:** **for** $j = 1, 2, \ldots, \lceil\sqrt{m}\rceil$ **do**
**4:** **if** $i \neq j$ **then**
**5:** $\{R_1, \ldots, R_{n'}\} := FirstFit(R(\pi_i, \pi_j), c, m)$
**6:** **else**
**7:** $\{R_1, \ldots, R_{n'}\} := FirstFit(R(\pi_i, \pi_i) - \bigcup_{\sigma\in\pi_i}\{R(\sigma^+, \sigma^+) \cup R(\sigma^-, \sigma^-)\}, c, m)$
**8:** **end**; /* if */
**9:** **for** $k = 1, 2, \ldots, n'$ **do**
**10:** Route $\sigma'$ follows cycles in $\pi_i$ in order to pick up requests in $R_k$;
**11:** Route $\sigma'$ follows cycles in $\pi_j$ in order to deliver requests in $R_k$
**12:** **end** /* for */
**13:** **end** /* for */
**14:** **end**; /* for */
**15:** Route $\sigma'$ follows all cycles in $\pi$ to serve requests in $\bigcup_{\sigma\in\pi}\{R(\sigma^+, \sigma^+) \cup R(\sigma^-, \sigma^-)\}$;
**16:** Route $\sigma'$ returns to depot $q$ by following path $\sigma^-$.

In Line 10, route $\sigma'$ is represented by $\sigma' := \sigma' \oplus t \cup \sigma^-_{i,1} \oplus p_{i,1} \oplus \sigma^+_{i,1} \oplus t \oplus \sigma^-_{i,2} \oplus p_{i,2} \oplus \sigma^+_{i,2} \oplus t \oplus \cdots \oplus t \oplus \sigma^-_{i,\lceil\sqrt{m}\rceil} \oplus p_{i,\lceil\sqrt{m}\rceil} \oplus \sigma^+_{i,\lceil\sqrt{m}\rceil} \oplus t$. (Note that if route $\sigma_{i,j}$ does not include a depot, then

let $\sigma^+_{i,j} = \sigma_{i,j} - t$, $\sigma^-_{i,j} = \emptyset$, and $p_{i,j} = \emptyset$). Line 11 is executed in the same way. In Line 15, route $\sigma'$ is represented by $\sigma' := \sigma' \oplus t \oplus \sigma^-_{1,1} \oplus p_{1,1} \oplus \sigma^+_{1,1} \oplus t \oplus \sigma^-_{1,2} \oplus p_{1,2} \oplus \sigma^+_{1,2} \oplus t \oplus \cdots \oplus t \oplus \sigma^-_{\lceil\sqrt{m}\rceil,\lceil\sqrt{m}\rceil} \oplus p_{\lceil\sqrt{m}\rceil,\lceil\sqrt{m}\rceil} \oplus \sigma^+_{\lceil\sqrt{m}\rceil,\lceil\sqrt{m}\rceil} \oplus t$. As the same with Lemma 4.2, the following lemma holds.

**Lemma 4.4.** *PDP route obtained by* CONVERT2 *serves all requests in $R$.*

**Proof:** By iterating Line 4-12 for $i, j = 1, \ldots, \lceil\sqrt{m}\rceil$, all requests in $R - \bigcup_{\sigma\in\pi}\{R(\sigma^+, \sigma^+) \cup R(\sigma^-, \sigma^-)\}$ are served. All requests in $\bigcup_{\sigma\in\pi}\{R(\sigma^+, \sigma^+) \cup R(\sigma^-, \sigma^-)\}$ are served in Line 15. Thus we have the lemma. □

We analyze travel cost of routes constructed by CONVERT2. Lemma 4.3 also holds for CONVERT2.

**Lemma 4.5.** *For given $i \in [1, \lceil\sqrt{m}\rceil]$, let $d'_i$ be the travel cost that is required to follow cycles in $\pi_i$ in Line 10 of* CONVERT2 *by iterating for all $j = 1, \ldots, \lceil\sqrt{m}\rceil$. Then it holds*

$$d'_i < 3\lceil\sqrt{m}\rceil \cdot d(\pi_i).$$

**Proof:** Omitted since it can be traced as in Lemma 4.3. □

We are now ready to prove Theorem 4.3.

**Proof of Theorem 4.3:** Let $d'$ be the travel cost of a subsequence of route $\sigma'$ that is constructed in Line 2-15 in CONVERT2. Since travel cost $d'$ is the same with the travel cost of routes that is constructed in CONVERT, it holds

$$d' < 6\lceil\sqrt{m}\rceil \cdot \sum_{i=1}^{\lceil\sqrt{m}\rceil} d(\pi_i) + d(\pi) \tag{4.4}$$

by using the proof of Theorem 4.2. Since travel cost of a subsequence of route $\sigma'$ that is constructed in Line 1 and 16 in CONVERT2 is less than or equal to $d(\pi)$, it holds

$$opt_{\mathrm{PDP}}(I) \quad < \quad 6\lceil\sqrt{m}\rceil \cdot \sum_{i=1}^{\lceil\sqrt{m}\rceil} d(\pi_i) + 2d(\pi).$$

By applying (4.2), it holds

$$opt_{\mathrm{PDP}}(I) \quad < \quad (6\lceil\sqrt{m}\rceil + 2) \cdot opt_{\mathrm{PDPT}}(I).$$

□

If we use the number $p$ of requests, the next theorem holds by $m \leq p$.

**Theorem 4.5.** *Suppose that each vehicle can visit the transshipment point any number of times. Let $I = (Q, R, c, t)$ be an instance with $p$ requests. Then it holds*

$$opt_{\mathrm{PDP}}(I) < (6\lceil\sqrt{p}\rceil + 2) \cdot opt_{\mathrm{PDPT}}(I).$$

□

## 4.5 Lower bound on travel cost of PDPC solutions to PDPTC solutions

We next introduce an instance where bound $O(\sqrt{m})$ is tight for a special case that all pickup points are required to be visited before delivery points in each route. The constraint corresponds to constraint (f) in Chapter 1. This section presents an instance $I'$ that satisfies $opt_{\mathrm{PDPC}}(I') = \sqrt{m} \cdot opt_{\mathrm{PDPTC}}(I')$. In real logistics, pickup operations are usually held through the night since many factories (pickup points) are open for 24 hours, while delivery operations are held in the daytime since stores (delivery points) are open only in the daytime. Therefore the assumption that all pickup points are required to be visited before delivery points holds for many cases.

**Theorem 4.6.** *Let $m$ be the number of cycles in a PDPTC solution. Then there exists an instance $I'$ that satisfies*

$$opt_{\mathrm{PDPC}}(I') = \sqrt{m} \cdot opt_{\mathrm{PDPTC}}(I').$$

**Proof:** Let instance $I'$ consist of a complete graph with depot set $Q = \{q_1, q_2, \ldots, q_m\}$, a transshipment point $t$, and a customer set $\{r_1^+, r_2^+, \ldots, r_m^+, r_1^-, r_2^-, \ldots, r_m^-\}$, where $r_i^+$ (resp., $r_i^-$), $i = 1, \ldots, m$, are pickup (resp., delivery) points. Let set $R$ of requests in $I'$ be $R = \bigcup_i \bigcup_j \{r_i^+, r_j^-\}$, and let $q(r) = 1$ for all $r \in R$. Suppose that distance $d$ is symmetric, and it hold $d(r_i^+, r_j^-) = d(r_i^+, r_j^+) = d(r_i^-, r_j^-) = d(p_i, r_j^-) = 2$ and $d(p_i, r_j^+) = 0$ for all $i, j = 1, \ldots, m$. Furthermore, let $d(r_i^+, t) = d(r_i^-, t) = 1$ for $i = 1, \ldots, m$. Let $c = m$. Note that the number $p$ of requests is equal to $m^2$.

In a PDPTC solution, a vehicle that is stationed at $q_i$, $i = 1, 2, \ldots, m$, picks up requests in $\bigcup_{j=1}^m \{r_i^+, r_j^-\}$ at $r_i^+$, and drops requests in $\bigcup_{j=1, j\neq i}^m \{r_i^+, r_j^-\}$ at transshipment point $t$. Next, it picks up requests in $\bigcup_{j=1, j\neq i}^m \{r_j^+, r_i^-\}$, which are delivered to transshipment point $t$ in advance by other vehicles that are stationed at $q_j$ ($j \neq i$). After the vehicle delivers requests in $\bigcup_{j=1}^m \{r_j^+, r_i^-\}$ to customer $r_i^-$, it finally returns to depot $q_i$. The total travel cost for the solution is $4m$. Fig 4.3 illustrates the PDPTC solution.

We next consider a PDPC solution. In order to give a PDPC solution, we divide set $C$ of customers into $2\sqrt{m}$ subsets and renumber indices such that $C_i^+ = \{r_{i,1}^+, r_{i,2}^+, \ldots, r_{i,\sqrt{m}}^+\}$ and $C_i^- = \{r_{i,1}^-, r_{i,2}^-, \ldots, r_{i,\sqrt{m}}^-\}$ for $i = 1, \ldots, \sqrt{m}$. Let $R(C_i^+, C_j^-)$ denote the set of requests whose pickup points belong to $C_i^+$ and delivery points belong to $C_j^-$. For each $i, j = 1, \ldots, \sqrt{m}$,

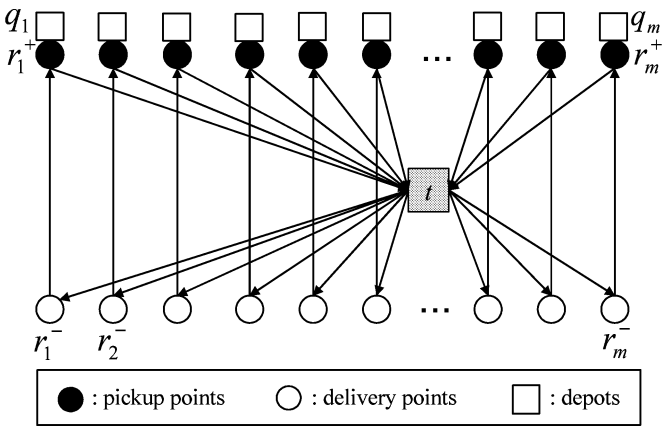



Figure 4.3: A PDPTC solution for instance $I'$.

a vehicle picks up requests in $R(C_i^+, C_j^-)$ by visiting vertices in $C_i^+$, and delivers them by visiting vertices in $C_j^-$. Since travel cost for a route is $4\sqrt{m}$ and the number of routes is $\sqrt{m} \times \sqrt{m} = m$, the total travel cost for the solution is $4\sqrt{m} \times m = 4m\sqrt{m}$. In order to prove the tightness, we show that there exists no PDPC solution whose travel cost is less than $4m\sqrt{m}$. Consider a route that serves $m$ requests in a PDPC solution. Suppose that the route includes $m_1 (\leq m)$ pickup points and $m_2 (\leq m)$ delivery points, where $m_1 \times m_2 \geq m$ holds. The travel cost for the route is $2(m_1 + m_2)$, which is the minimum when $m_1 = m_2 = \sqrt{m}$ under the restriction that $m_1 \times m_2 \geq m$. Hence the minimum travel cost for the route that serves $m$ requests is $4\sqrt{m}$. Since the number of requests is $m^2$, the total travel cost is $4m\sqrt{m}$. Thus the bound is shown to be $4m\sqrt{m}/(4m) = \sqrt{m}$. □

# Chapter 5

# Worst-Case Analysis on the Optimal Cost between PDP and Multi-Trip PDP with Consecutive Pickups and Deliveries

## 5.1 Introduction

In this chapter, we consider the multi-trip PDP with consecutive pickups and deliveries (PDPCMT). In PDPCMT, any vehicle which has begun a delivery action is not allowed to take pickup actions until all of the loads on the vehicle are delivered. In this chapter, we are interested in how the least travel cost can be increased by the additional requirement, and examine the maximum ratio of the optimal value of PDPCMT to that of PDP over all instances with $p$ requests. We show that the maximum ratio is bounded from above by $O(\log p)$ and from below by $\Omega(\log p / \log \log p)$. In order to show the lower bound, we introduce a partition problem that asks to find a partition of vertices on a binary tree, and present a lower bound on the partition problem.

## 5.2 Problem definition

This section defines PDP and PDPCMT that are introduced in Chapter 1 again, and gives additional definitions. Depots and customers to be visited by vehicles are represented by vertices in an edge-weighted complete digraph with a vertex set $V$, which is given by distance functions $d(u, v)$ for all ordered pairs of vertices $u$ and $v$. Distance $d(u, v)$ is a nonnegative real number, and in general asymmetric, i.e., $d(u, v) \neq d(v, u)$ may hold. Distance $\{d(u, v) \mid u, v \in V\}$ may not satisfy the Triangle Inequality.

An instance consists of a set $Q$ of *depots*, a set $R$ of *requests*, and a vehicle capacity $c > 0$. Each request $r \in R$ consists of a *pickup action* $r^+$, a *delivery action* $r^-$, and a quantity $q(r)$ of loads for the request. That is, quantity $q(r)$ of loads is required to be picked up at a specified vertex where $r^+$ is taken and to be delivered to a specified vertex where $r^-$ is taken. We may denote the vertex where request $r$ is picked up (resp., delivered) by $r^+$ (resp., $r^-$), and call the vertex $r^+$ a *pickup point* (resp., $r^-$ a *delivery point*). Let $A = \{r^+, r^- \mid r \in R\}$, i.e., the set of all actions for $R$.

Each vehicle must start from a depot in $Q$, and return to the same depot after serving some of the requests in $R$. A *route* is the requests assigned to a vehicle, and is represented by the sequence $\sigma = [a_0, a_1, a_2, \ldots, a_m, a_{m+1}]$ of actions of the requests in the order that the vehicle serves, where $a_1, a_2, \ldots, a_m \in A$ ($m$ is an even integer) and $a_0, a_{m+1} \in Q$. The travel cost of $\sigma$ is defined by

$$cost(\sigma) = \sum_{0 \le i \le m} d(a_i, a_{i+1}).$$

A *solution* $s$ is a set of routes that serves all requests in $R$, and its cost $cost(s)$ is defined by the sum of the travel costs of the routes $\sigma$ in $s$, that is, $cost(s) = \sum_{\sigma \in s} cost(\sigma)$. The objective is to find a minimum cost solution, and we assume that any number of vehicles is available.

We review PDP and PDPCMT below.

**Pickup and Delivery Problem (PDP):**
**Input:** An instance $I = (Q, R, c)$.
**Output:** A minimum cost solution that satisfies the following constraints:

(a) the total quantity of loads during a route does not exceed vehicle capacity $c$ at any time;

(b) each request $r$ is served by exactly one vehicle, and the actions $r^+$ and $r^-$ are taken only once by the vehicle;

(c) (*coupling constraint*) actions $r^+$ and $r^-$ of each request $r$ must appear in the same route, and no request $r$ is allowed to be temporarily dropped at any vertices; and

(d) (*precedence constraint*) for each request $r$, action $r^+$ must be taken before action $r^-$.

**Multi-Trip PDP with Consecutive Pickups and Deliveries (PDPCMT):**
**Input:** An instance $I = (Q, R, c)$.
**Output:** A minimum cost solution that satisfies constraints (a), (b), (c), (d), and the following constraint:

(e) once a delivery action begins, pickup actions cannot be taken until all of the loads on the vehicle are delivered.

A route satisfying constraint (e) is a sequence of subsequences such that first a vehicle with no loads on it takes pickup actions for some requests and takes delivery actions for these requests. We call such a subsequence a *trip*.

We next review PDPTCMT and PDPTC that are introduced in Chapter 1. The multi-Trip pickup and delivery problem with transfer with consecutive pickups and deliveries (PDPTCMT) is a problem whose output is a minimum cost solution that satisfies constraints (a), (b), (d) and (e). The pickup and delivery problem with transfer with consecutive pickups and deliveries (PDPTC) is a problem whose output is a minimum cost solution that satisfies constraints (a), (b), (d) and (f).

Given a set $Z \subseteq R$ of requests, let $Z^+$ (resp., $Z^-$) denote the set of pickup (resp., delivery) actions for $Z$. Given a solution $s$ to PDPCMT or PDP, we denote by $T(s)$ the set of trips in $s$. Let $R(t)$ denote the set of requests that are served on trip $t$. Given a route $\sigma$, let $A(\sigma)$ represent the set of actions along $\sigma$.

Given a request $r \in R$, we call $d(r^+, r^-)$ the *travel cost of request* $r$. Given a trip $t = [a_1, a_2, \ldots, a_n]$, let $cost(t) = \sum_{1 \le i \le n-1} d(a_i, a_{i+1})$.

Given a PDPCMT route $\sigma = [a_0, t_1, t_2, \ldots, t_k, a_1]$ with $a_0, a_1 \in Q$ and trips $t_1, t_2, \ldots, t_k$, we define the *loaded travel cost* as the travel cost with loads, i.e., $cost_f(\sigma) = \sum_{i=1}^{k} cost(t_k)$. Given a PDPCMT solution $s$, let $cost_f(s) = \sum_{\sigma \in s} cost_f(\sigma)$. In Fig. 1.1, bold lines correspond to subsequences with the loaded travel cost while dashed lines correspond to those with no loads.

## 5.3 Lower bound on a partition problem

This section derives a lower bound on a partition problem that asks to find a partition of vertices on a complete binary tree with the minimum cost. The lower bound will be used in Section 5.5 to show a lower bound on the maximum ratio of the optimal cost of PDPCMT to that of PDP.

### 5.3.1 Introduction of a partition problem

This subsection introduces a partition problem Let $h$ be a positive integer, and let $c = 2^h$, and let $\mathcal{T}_c = (V, E)$ denote a complete binary tree with $2^{c-1}$ leaves. Hence the number of vertices in $V$ is $|V| = \sum_{i=1}^{c} 2^{i-1} = 2^c - 1$. Fig 5.1 illustrates $\mathcal{T}_c$ with $c = 4$.

For a vertex $v \in V$, let $L(v)$ denote the set of leaves that are the descendants of vertex $v$ (including $v$). For a subset $X \subseteq V$, let $L(X)$ denote the set $\cup_{v \in X} L(v)$.

We now introduce a *partition problem of vertices on a binary tree* (PPBT). We call a partition $V_1, V_2, \ldots, V_n$ $(n \ge 1)$ of $V$ is *feasible* if it satisfies $V_1 \cup V_2 \cup \cdots \cup V_n = V$, $V_i \cap V_j = \emptyset$

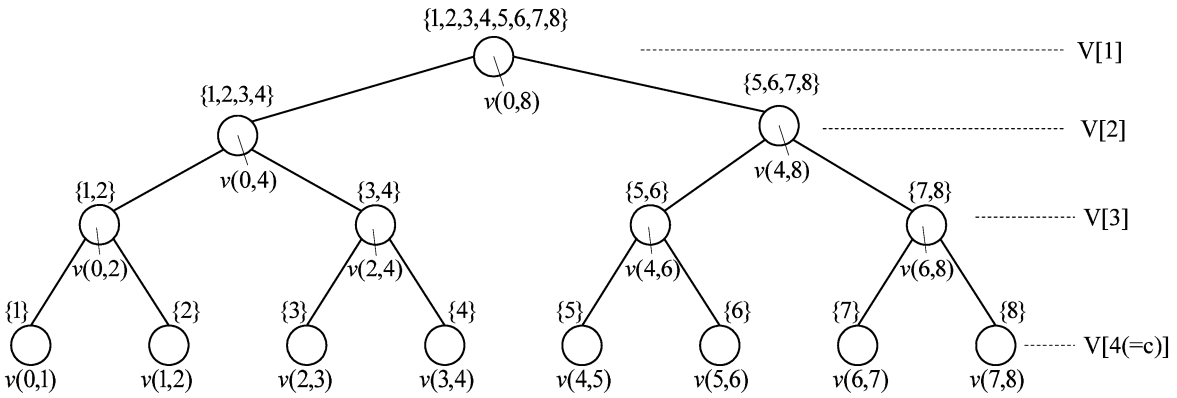



Figure 5.1: A complete binary tree with $c = 4$

for $1 \leq i < j \leq n$, and the number of vertices in each subset $V_i$ is less than or equal to $c$. We call a feasible partition *a solution* to PPBT. The cost $cost_L(s)$ of a solution $s = \{V_1, V_2, \ldots, V_n\}$ is defined to be the number of times that leaves appear in $L(X)$, $X \in s$, i.e.,

$$cost_L(s) = \sum_{X \in s} |L(X)|.$$

PPBT asks to find a solution $s$ that minimizes $cost_L(s)$. This section proves the following theorem.

**Theorem 5.1.** *For a complete binary tree $\mathcal{T}_c$ with $2^{c-1}$ leaves, any solution $s^*$ to PPBT satisfies $cost_L(s^*) \geq 2^{c-2} \cdot c/\log_2 c$. Furthermore there exists a solution $\hat{s}$ such that $cost_L(\hat{s}) = 2^{c-1} \cdot c/\log_2 c$.*

□

We introduce some notations on tree $\mathcal{T}_c$. We denote by $V[i]$, $i = 1, 2, \ldots, c$, the set of vertices on the $i$-th layer from the root vertex in $\mathcal{T}_c$. We denote set $V[c]$ of vertices on the $c$-th layer in $\mathcal{T}_c$ by $V[c] = \{v(0,1), v(1,2), v(2,3), \ldots, v(2^{c-1}-1, 2^{c-1})\}$. For $i = 1, 2, \ldots, c$, set $V[i]$ consists of $2^{i-1}$ vertices and is denoted by

$$V[i] = \{v((j-1) \cdot 2^{c-i}, j \cdot 2^{c-i}) \mid j = 1, 2, \ldots, 2^{i-1}\}. \tag{5.1}$$

The set $V[i]$ satisfies

$$|L(V[i])| = |L(v(0, 2^{c-1}))| = 2^{c-1}. \tag{5.2}$$

Fig. 5.2 illustrates an example of a solution $s$ to PPBT with $n = 4$. Circles with dashed lines show sets in solution $s$. The solution $s$ consists of four sets, $V_1 = \{v(0,8), v(0,4), v(0,2), v(1,2)\}$, $V_2 = \{v(0,1), v(4,5), v(4,6), v(4,8)\}$, $V_3 = \{v(2,4), v(2,3), v(3,4), v(5,6)\}$, and $V_4 = \{v(6,8), v(6,7), v(7,8)\}$

The partition $V_1, V_2, V_3$ and $V_4$ is a feasible partition and it hold $|L(V_1)| = |L(v(0,8))| = 8$, $|L(V_2)| = |L(v(0,1))|+|L(v(4,8))| = 1+4 = 5$, $|L(V_3)| = |L(v(2,4))|+|L(v(5,6))| = 2+1 = 3$, and $|L(V_4)| = |L(v(6,8))| = 2$.

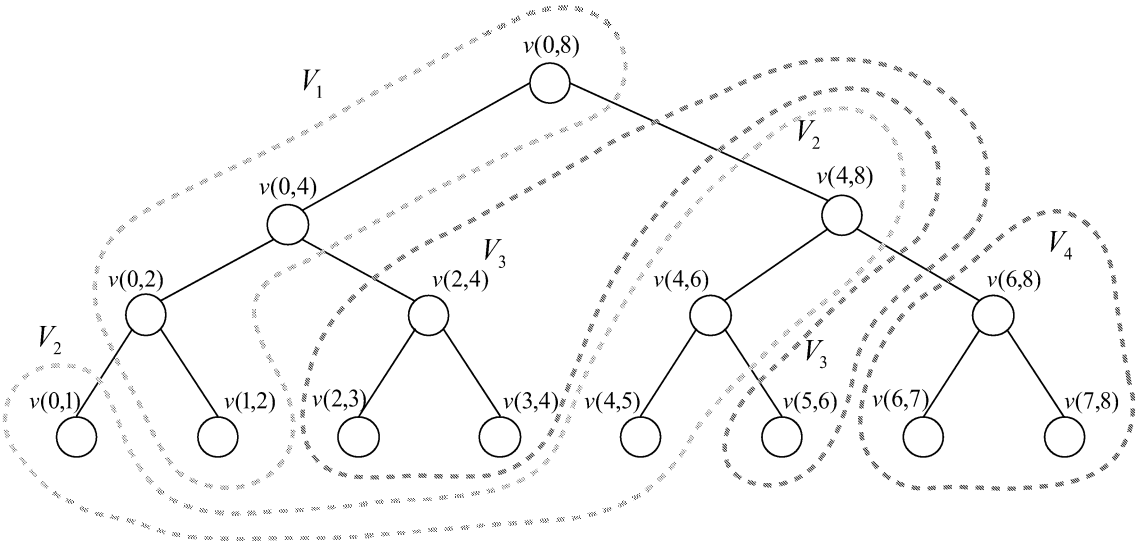



Figure 5.2: An example of a solution $s$ to PPBT with $c = 4$.

### 5.3.2 Definition of head vertices

This subsection gives some definitions that are used to analyze a lower bound on cost of an optimal solution to PPBT. Given a solution $s$ to PPBT and a vertex $v \in V$, we denote by $V(v; s)$ the set of vertices in solution $s$ to which vertex $v$ belongs.

Given a solution $s$ to PPBT, a vertex $v \in V$, and a vertex $v_1 \in V(v; s) - \{v\}$, we say that vertex $v$ *is covered by* vertex $v_1$ in solution $s$ if vertex $v$ is a descendant vertex of $v_1$ on tree $\mathcal{T}_c$, i.e., it holds $L(v) \subset L(v_1)$. We denote by $Cover(v; s)$ the set of vertices by which vertex $v$ is covered in solution $s$. For a vertex $v \in V$, $Cover(v; s)$ is formally described by

$$Cover(v; s) = \{v_1 \in V(v; s) - \{v\} \mid L(v) \subset L(v_1)\}.$$

For the solution $s$ shown in Fig. 5.2, we have $Cover(v(1,2); s) = \{v(0,8), v(0,4), v(0,2)\}$, $Cover(v(0,2); s) = \{v(0,8), v(0,4)\}$, $Cover(v(0,4); s) = \{v(0,8)\}$, $Cover(v(2,3); s) = Cover(v(3,4); s) = \{v(2,4)\}$, $Cover(v(0,8); s) = Cover(v(0,1); s) = Cover(v(2,4); s) = Cover(v(4,8); s) = Cover(v(5,6); s) = Cover(v(6,8); s) = \emptyset$.

We next introduce *head vertices* of a solution $s$, which attain $cost_L(s)$ of solution $s$ to PPBT. For a set $X$ in a solution $s$ to PPBT, a vertex $v \in X$ is called a *head vertex* in $X$ if $X$ contains no vertex that covers $v$, i.e., $Cover(v; s) = \emptyset$. Let $Head(X)$ denote the set of head

vertices in $X$. We easily see that for any set $X \in s$ it holds

$$\bigcup_{v \in Head(X)} L(v) = \bigcup_{v \in X} L(v) = L(X). \tag{5.3}$$

For the solution $s$ shown in Fig. 5.2, we have $Head(V_1) = \{v(0,8)\}$, $Head(V_2) = \{v(0,1), v(4,8)\}$, $Head(V_3) = \{v(2,4), v(5,6)\}$, and $Head(V_4) = \{v(6,8)\}$.

Given a solution $s$, let $Head(s) = \bigcup_{X \in s} Head(X)$. We obtain the following lemma.

**Lemma 5.1.** *Let $s$ be a solution to PPBT. Then it holds*

$$cost_L(s) = |L(Head(s))|.$$

**Proof:** Let $V_1, V_2, \ldots, V_n$ be the sets of vertices in $s$. By (5.3), we have $L(V_i) = \bigcup_{v \in Head(V_i)} L(v)$, $i = 1, 2, \ldots, n$. Since no vertex appears as head vertices for two distinct sets $V_i, V_j \in s$, we have $cost_L(s) = \sum_{V_i \in s} |L(V_i)| = \sum_{V_i \in s} \sum_{v \in Head(V_i)} |L(v)| = \sum_{v \in Head(s)} |L(v)| = |L(Head(s))|$. □

### 5.3.3 Definition of a solution with a systematic structure

This subsection introduces a certain solution $\hat{s}$ to PPBT, which has a systematic structure, and is used to analyze a lower bound on the optimal cost of PPBT. Recall that $c = 2^h$ holds for binary tree $\mathcal{T}_c$. We partition the vertex set $V$ of $\mathcal{T}_c$ into subsets $V[k, j]$, $k = 1, \ldots, c/h$, $m = 1, \ldots, 2^{(k-1)h}$ such that each set $V[k, j]$ induces a complete binary subtree with exactly $c - 1$ vertices from $\mathcal{T}_c$, as shown in Fig. 5.3. We then define solution $\hat{s}$ by

$$\hat{s} = \{V[k, j] \mid k = 1, 2, \ldots, c/h, j = 1, 2, \ldots, 2^{(k-1)h}\},$$

where the highest vertex in $V[k, j]$ is the unique head vertex in $V[k, j]$. Boxes with dashed lines in Fig. 5.3 show sets $V[k, j]$, $k = 1, 2$, $j = 1, 2, \ldots, 2^{2(k-1)}$, of vertices with $c = 4$ and $h = 2$. Note that $v(0, 8)$ is the head vertex in $V[1, 1]$

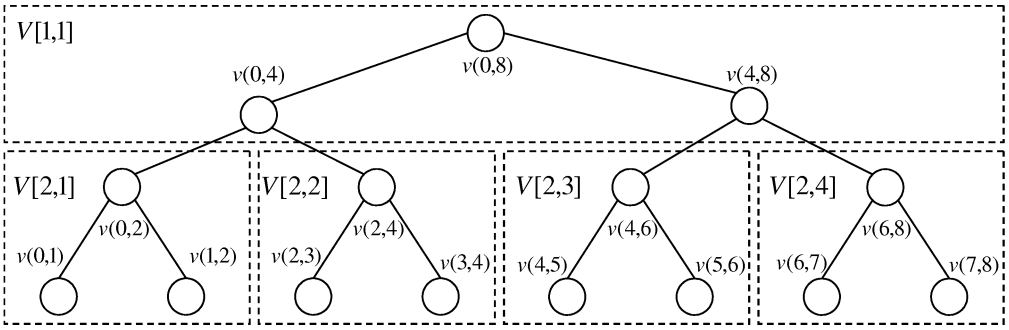



Figure 5.3: A solution $\hat{s}$ with $c = 4$ and $h = 2$.

More formally sets $V[k,j]$ for $k = 1,2,\ldots,c/h$ and $j = 1,2,\ldots,2^{(k-1)h}$ are defined as follows. The first set $V[1,1]$ consists of the vertices in the top $h$ layers of $\mathcal{T}_c$, i.e., $V[1,1] = V[1] \cup V[2] \cup \cdots \cup V[h]$. The number of vertices in $V[1,1]$ is $\sum_{i=1}^{h} 2^{i-1} = 2^h - 1 = c - 1$.

For each $k = 1,2,\ldots,c/h$, consider the set $V[(k-1)h+1] \cup V[(k-1)h+2] \cup \cdots \cup V[kh]$ in the $k$th $h$ layers of $\mathcal{T}_c$. Recall from (5.1) that

$$V[(k-1)h+1] = \{v((j-1)\cdot 2^{c-(k-1)h-1}, j\cdot 2^{c-(k-1)h-1}) \mid j = 1,2,\ldots,2^{(k-1)h}\},$$

and $|V[(k-1)h+1]| = 2^{(k-1)h}$. We partition $V[(k-1)h+1] \cup V[(k-1)h+2] \cup \cdots \cup V[kh]$ into sets $V[k,j]$, $j = 1,2,\ldots,2^{(k-1)h}$ from left to right, as shown in Fig. 5.3, so that the $j$th set $V[k,j]$ in the $k$th $h$ layers contains

- the $j$th vertex $v((j-1)\cdot 2^{c-(k-1)h-1}, j\cdot 2^{c-(k-1)h-1})$ in $V[(k-1)h+1]$ as its head vertex, and
- the vertices in $V[(k-1)h+1] \cup V[(k-1)h+2] \cup \cdots \cup V[kh]$ that are covered by the head vertex.

Thus $V[k,j]$ is expressed by

$$\begin{aligned} V[k,j] \;=\; & \{v \in V[(k-1)h+1] \cup V[(k-1)h+2] \cup \cdots \cup V[kh] \\ & \mid L(v) \subseteq L(v((j-1)\cdot 2^{c-(k-1)h-1}, j\cdot 2^{c-(k-1)h-1}))\}. \end{aligned} \tag{5.4}$$

We easily see that each $V[k,j]$ contains exactly $c-1$ vertices. In the example of Fig. 5.3, solution $\hat{s}$ is given by sets $V[1,1] = \{v(0,8), v(0,4), v(4,8)\}$, $V[2,1] = \{v(0,2), v(0,1), v(1,2)\}$, $V[2,2] = \{v(2,4), v(2,3), v(3,4)\}$, $V[2,3] = \{v(4,6), v(4,5), v(5,6)\}$, and $V[2,4] = \{v(6,8), v(6,7), v(7,8)\}$. The head vertices in $\hat{s}$ in Fig. 5.4 are $v(0,8), v(0,2), v(2,4), v(4,6)$, and $v(6,8)$.

By using (5.2), for each $k = 1,2,\ldots,c/h$, the head vertices in $V[k,j]$, $j = 1,2,\ldots,2^{(k-1)h}$, satisfy

$$\begin{aligned} \sum_{j=1}^{2^{(k-1)h}} |L(V[k,j])| &= \sum_{j=1}^{2^{(k-1)h}} |L(v((j-1)\cdot 2^{c-(k-1)h-1}, j\cdot 2^{c-(k-1)h-1}))| \\ &= |L(v(0,2^{c-1}))| \\ &= 2^{c-1}. \end{aligned} \tag{5.5}$$

Thus the head vertex in $V[k,j]$ satisfies

$$\begin{aligned} |L(V[k,j])| &= |L(v((j-1)\cdot 2^{c-(k-1)h-1}, j\cdot 2^{c-(k-1)h-1}))| \\ &= 2^{c-1}/2^{(k-1)h} \\ &= 2^{c-(k-1)h-1}. \end{aligned} \tag{5.6}$$

By summing up (5.5) for $k = 1, 2, \ldots, c/h$, we have

$$\begin{aligned} cost_L(\hat{s}) &= |L(Head(\hat{s}))| \\ &= 2^{c-1} \cdot c/h \\ &= 2^{c-1} \cdot c/\log_2 c. \end{aligned} \tag{5.7}$$

### 5.3.4 Analyzing a lower bound on PPBT

Let $s^*$ be a solution to PPBT. This subsection analyzes a lower bound on $cost_L(s^*)$ by comparing $s^*$ with solution $\hat{s}$. For this, we first partition the set $V$ of vertices in tree $\mathcal{T}_c$ into five sets $Z_1, Z_2, Z_2', Z_3$, and $Z_4$ according to two solutions $\hat{s}$ and $s^*$ as follows.

$$Z_1 = \{v \mid v \in Head(\hat{s}), v \in Head(s^*)\}, \tag{5.8}$$
$$Z_2 = \{v \mid v \in Head(\hat{s}), v \notin Head(s^*), Cover(v; s^*) \cap Head(\hat{s}) = \emptyset\}, \tag{5.9}$$
$$Z_2' = \{v \mid v \in Head(\hat{s}), v \notin Head(s^*), Cover(v; s^*) \cap Head(\hat{s}) \neq \emptyset\}, \tag{5.10}$$
$$Z_3 = \{v \mid v \notin Head(\hat{s}), v \in Head(s^*)\}, \tag{5.11}$$
$$Z_4 = \{v \mid v \notin Head(\hat{s}), v \notin Head(s^*)\}. \tag{5.12}$$

Note that $V = Z_1 \cup Z_2 \cup Z_2' \cup Z_3 \cup Z_4$, $Z_i \cap Z_j = \emptyset$ ($i, j \in \{1, 2, 3, 4\}$ and $i \neq j$), and $Z_i \cap Z_2' = \emptyset$ ($i \in \{1, 2, 3, 4\}$).

Fig. 5.4 illustrates an examples of sets $V_1 = \{v(0,1), v(0,2), v(0,4), v(4,6)\}$ and $V_2 = \{v(0,8), v(4,8), v(2,4), v(7,8)\}$ of vertices in a solution $s^* = \{V_1, V_2, V_3, V_4\}$, where we omit drawing the other sets $V_3, V_4 \in s^*$ for simplicity. Each box with dashed lines corresponds to a set in $\hat{s}$ that is shown in Fig. 5.3. For set $V_1$, vertices $v(0,4)$ and $v(4,6)$ are the head vertices in $V_1$, and we have $Z_1 \cap V_1 = \{v(4,6)\}, Z_2 \cap V_1 = \{v(0,4)\}, Z_2' \cap V_1 = \emptyset, Z_3 \cap V_1 = \{v(0,2)\}$, and $Z_4 \cap V_1 = \{v(0,1)\}$. For set $V_2$, vertex $v(0,8)$ is the head vertex in $V_2$, and we have $Z_1 \cap V_2 = \{v(0,8)\}, Z_2 \cap V_2 = \emptyset, Z_2' \cap V_2 = \{v(2,4)\}, Z_3 \cap V_2 = \emptyset$, and $Z_4 \cap V_2 = \{v(4,8), v(7,8)\}$. We obtain the following lemma.

**Lemma 5.2.** *The sets $Z_1, Z_2, Z_2'$, and $Z_3$ of vertices defined in (5.8) - (5.11) satisfy*
(i) $|L(Z_1)| + |L(Z_2)| + |L(Z_2')| = 2^{c-1} \cdot c/h$;
(ii) $|L(Z_2)| \leq |L(Z_3)|$; *and*
(iii) $|L(Z_2')| \leq |L(Z_1)| + |L(Z_2)|$.

**Proof:** (i) The equality follows from $Z_1 \cup Z_2 \cup Z_2' = Head(s^*)$ and (5.7).
(ii) Let $v_2$ be an arbitrary vertex in $Z_2$. Then there exists a vertex $v_3 \in Z_3 \cap Cover(v_2; s^*)$, since $v_2$ is not a head vertex in $V(v_2; s^*)$, and is covered by a head vertex of $V(v_2; s^*)$ in

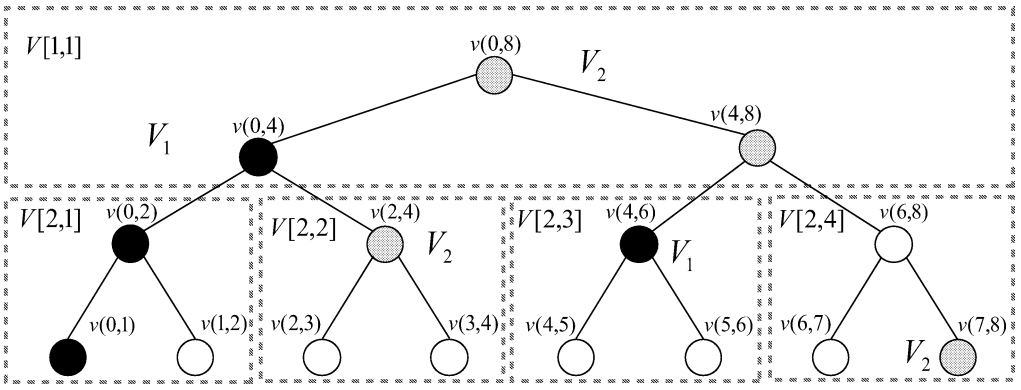



Figure 5.4: Tree $\mathcal{T}_c$ with A solution $\hat{s}$ with $c = 4$ and $h = 2$, where the each set in solution $\hat{s}$ is denoted by a box with dashed lines, and the vertices in sets $V_1$ and $V_2$ in solution $s^*$ are depicted by black and grey circles, respectively.

$s^*$. Notice that it holds $L(v_2) \subset L(v_3)$ since vertex $v_2$ is covered by vertex $v_3$ in solution $s^*$. Hence $L(Z_2) \subseteq L(Z_3)$ holds.
(iii) Let $X \in s^*$ be an arbitrary set with $X \cap Z_2' \neq \emptyset$. For each vertex $u_2 \in X \cap Z_2'$, it holds $Cover(u_2; s^*) \cap Head(\hat{s}) \neq \emptyset$ by (5.10), and hence $Cover(u_2; s^*) \cap Head(\hat{s}) = Cover(u_2; s^*) \cap (Z_1 \cup Z_2 \cup Z_2') \neq \emptyset$. We choose a highest vertex $v_1 \in Cover(u_2; s^*) \cap (Z_1 \cup Z_2 \cup Z_2')$ in tree $\mathcal{T}_c$. Then $v_1 \in Z_1 \cup Z_2$ holds, since $v_1 \in Z_2'$ would imply that there exists a vertex $v' \in Cover(v_1; s^*) \cap Head(\hat{s})$, which covers $v_1$ and also belongs to $Cover(u_2; s^*) \cap Head(\hat{s})$, a contradiction to the choice of $v_1$ (since $v'$ is higher than $v_1$ in $\mathcal{T}_c$).

By the structure of solution $\hat{s}$, $v_1 \in Head(\hat{s})$ is the head vertex in a set $V[k_1, j_1] \in \hat{s}$ with $k_1 \in \{1, 2, \ldots, c/h - 1\}$ and $j_1 \in \{1, 2, \ldots, 2^{(k_1-1)h}\}$, and $u_2 \in Head(\hat{s})$ is the head vertex in a set $V[k_2, j_2] \in \hat{s}$ with $k_2 \in \{k_1 + 1, \ldots, c/h - 1\}$ and $j_2 \in \{1, 2, \ldots, 2^{(k_2-1)h}\}$. By (5.6), we have $|L(v_1)| = 2^{c-(k_1-1)h-1}$ and $|L(u_2)| = 2^{c-(k_2-1)h-1}$. Since $k_2 \geq k_1 + 1$, this implies

$$|L(u_2)| \leq |L(v_1)|/2^h.$$

Note that there may exist more than one vertex in $Z_2' \cap X$. By summing up the inequality over all vertices $u \in Z_2' \cap X$, we have

$$\sum_{u \in Z_2' \cap X} |L(u)| \leq \frac{|Z_2' \cap X|}{2^h} |L(v_1)| \leq |L(v_1)|,$$

where the second inequality follows from $|Z_2' \cap X| \leq c = 2^h$ by the feasibility of solution $s^*$. Note that $v_1 \in Cover(u_2; s^*) \subseteq V(u_2; s^*) = X$ holds. Therefore, by summing up the inequality over all sets in solution $s^*$, we have

$$\sum_{u \in Z_2'} |L(u)| \leq \sum_{v \in Z_1 \cup Z_2} |L(v)|,$$

as required.

We are ready to prove Theorem 5.1.

**Proof of Theorem 5.1:** By applying Lemma 5.2 (ii), (iii) and (i), we have

$$\begin{aligned} cost_L(s^*) &= |L(Head(s^*))| \\ &= |L(Z_1)| + |L(Z_3)| \\ &\geq |L(Z_1)| + |L(Z_2)| \\ &\geq \frac{1}{2}(|L(Z_1)| + |L(Z_2)| + |L(Z_2')|) \\ &= 2^{c-2} \cdot c/h. \end{aligned}$$

This and (5.7) prove Theorem 5.1. □

## 5.4 Upper bound on travel cost of PDPCMT solutions

This section derives the following upper bound on the maximum ratio of the optimal cost of PDPCMT to that of PDP.

**Theorem 5.2.** *Let $I = (Q, R, c)$ be an instance with $p \geq 2$ requests. Then it holds*

$$opt_{\text{PDPCMT}}(I) \leq \lceil \log_2 2p \rceil \cdot opt_{\text{PDP}}(I).$$

□

For this, we show how to convert a PDP solution to a PDPCMT solution.

**Theorem 5.3.** *Given a PDP route $\sigma$ to an instance $I$, PDPCMT routes $\pi_1, \pi_2, \ldots, \pi_k$ with*

$$\bigcup_{i=1}^{k} A(\pi_i) = A(\sigma) \text{ and } \sum_{i=1}^{k} cost(\pi_i) \leq \lceil \log_2 |A(\sigma)| \rceil \cdot cost(\sigma)$$

*can be constructed in polynomial time, where $|A(\sigma)|/2$ means the number of requests served in route $\sigma$.* □

Theorem 5.2 is obtained by applying Theorem 5.3 to each route $\sigma$ in an optimal PDP solution $s$ to instance $I$.

In this section, we denote a given PDP route by $\sigma = [a_0, a_1, a_2, \ldots, a_m, a_{m+1}]$, where $a_0, a_{m+1} \in Q$, $a_1, \ldots, a_m \in A$ and $m = 2p$. For notational simplicity, we assume that $m = 2^k$ holds for some integer $k \geq 1$ by introducing fictitious requests $r$ with $q(r) = 0$ and $r^- = r^+$ on vertex $a_h$ if necessary. We describe an algorithm that converts a PDP route $\sigma = [a_0, a_1, \ldots, a_m, a_{m+1}]$ to $k$ PDPCMT routes $\pi_1, \pi_2, \ldots, \pi_k$. Each route $\pi_j$ visits the vertices in route $\sigma$ in the same order serving a set $R_j$ of requests defined as follows.

We divide the set $A(1,1) = \{a_i \mid i = 1, 2, \ldots, 2^k\}$ of all actions in route $\sigma$ into two subsets $A(2,1) = \{a_i \mid i = 1, 2, \ldots, 2^{k-1}\}$ and $A(2,2) = \{a_i \mid i = 2^{k-1} + 1, 2^{k-1} + 2, \ldots, 2^k\}$ in the second level. By repeating this recursively, we define $2^{j-1}$ subsets at the $j$-th level to be

$$A(j,b) = \{a_i \mid i = (b-1)2^{k-j+1} + 1, (b-1)2^{k-j+1} + 2, \ldots, b2^{k-j+1}\}$$

for $b = 1, 2, \ldots, 2^{j-1}$, where $j = 1, 2, \ldots, k+1$. A request $r \in R$ is at the $j$-th level if $j$ is the smallest level such that a subset $A(j,b)$ contains both actions $r^+$ and $r^-$. Let us denote the subsets of requests at the $j$-th level by

$$R(j,b) = \{r \in R \mid r^+ \in A(j+1, 2b-1), r^- \in A(j+1, 2b)\},$$

$j = 1, 2, \ldots, k$ and $b = 1, 2, \ldots, 2^{j-1}$. We assign to route $\pi_j$ the requests in

$$R_j = R(j,1) \cup R(j,2) \cup \cdots \cup R(j, 2^{j-1}),$$

and let a vehicle serve the requests $R(j,1)$, $R(j,2), \ldots, R(j,2^{j-1})$ in this order along route $\pi_j$. Note that it holds $cost(\pi_j) = cost(\sigma)$ if $R_j \neq \emptyset$, $cost(\pi_j) = 0$ otherwise, for each $j = 1, 2, \ldots, k$.

The following lemma ensures the feasibility of routes $\pi_1, \pi_2, \ldots, \pi_k$.

**Lemma 5.3.** *Each routes $\pi_j$, $j = 1, \ldots, k$, is a PDPCMT routes, that is, route $\pi_j$ satisfies constraints (a),(b),(c),(d), and (e).*

**Proof:** Route $\pi_j$ satisfies constraint (c) since requests in $R(j,b)$ are served by one vehicle for each $j$ and $b$ and satisfies constraint (d) since pickup actions in $R(j,b)^+$ are taken before delivery actions in $R(j,b)^-$ for each $j$ and $b$. Since it holds $R_j \cap R_{j'} = \emptyset$ for each $j \neq j'$, route $\pi_j$ satisfies constraint (b). Route $\pi_j$ satisfies constraint (e), that is, actions in $R(j,b)^-$ precedes actions in $R(j,b+1)^+$ since $R(j,b)^- = A(j+1, 2b)$ and $R(j,b+1)^+ = A(j+1, 2b+1)$. We finally show that route $\pi_j$ satisfies constraint (a). When a vehicle finishes picking up requests in $R(j,b)$, the loads on the vehicle is the maximum during serving requests in $R(j,b)$. Since all the pickup actions in $R(j,b)^+$ precedes the delivery actions in $R(j,b)^-$ on PDP route $\sigma$, it holds $\sum_{r \in R(j,b)} q(r) \leq c$ for each $j$ and $b$. Thus we have the lemma. □

We are now ready to prove Theorem 5.3.

**Proof of Theorem 4.2:** Since $m = 2^k$, $m = |A(\sigma)|$, and $k$ is an integer number, we have $k = \lceil \log |A(\sigma)| \rceil$. For PDPCMT routes $\pi_j$, $j = 1, 2, \ldots, k$, constructed by converting a PDP route $\sigma$, it holds $cost(\pi_j) = cost(\sigma)$ or $cost(\pi_j) = 0$ for $j = 1, 2, \ldots, k$. Then we have

$$\sum_{j=1}^{k} cost(\pi_j) \leq k \cdot cost(\sigma) = \lceil \log |A(\sigma)| \rceil \cdot cost(\sigma).$$

Thus we have the theorem. □

## 5.5 Lower bound on travel cost of PDPCMT solutions

This section proves that the upper bound in Theorem 5.2 is tight up to factor of $O(\log \log p)$ by identifying such an instance for any large $p$. Thus we prove the next result.

**Theorem 5.4.** *For any integer $h \geq 1$, there exists an instance $I = (Q, R, c)$ with $p = 2^{2^h} - 1$ requests that satisfies*

$$opt_{\mathrm{PDPCMT}}(I) \geq \frac{\log_2(p+1)}{4 \log_2 \log_2(p+1)} \cdot opt_{\mathrm{PDP}}(I).$$

□

Such an instance $I$ in Theorem 5.4, which we denote by $G(p, \lambda)$, is defined below. Given an integer $p = 2^{2^h} - 1$ ($h \geq 1$) and a real $\lambda > 0$, let capacity be

$$c = 2^h = \log_2(p+1),$$

and let $V = \{v(0), v(1), v(2), \ldots, v(2^{c-1})\}$ be the set of vertices in $G(p, \lambda)$. We assume that all vertices in $V$ are arranged uniformly on a horizontal straight line-segment of length $\lambda$ from left to right as shown in Fig. 5.5. Hence $d(v(0), v(2^{c-1})) = \lambda$ and $d(v(i), v(i+1)) = \lambda/2^{c-1}$ for $i = 0, 1, \ldots, 2^{c-1} - 1$.

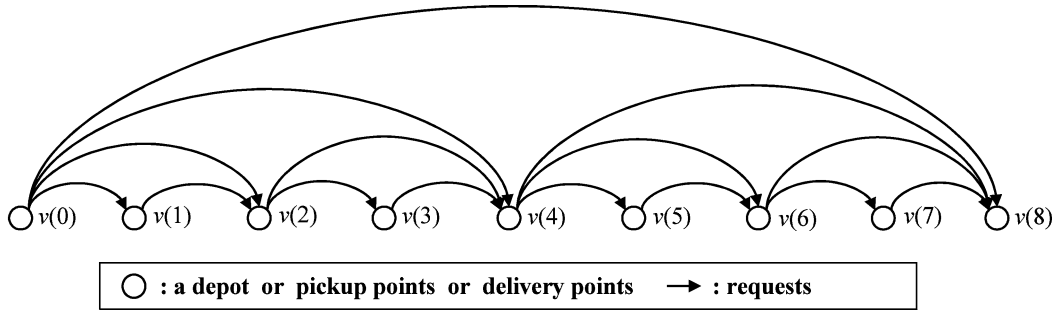



Figure 5.5: An instance $G(p, \lambda)$ with $c = 4$, $h = 2$ and $p = 15$.

Let $Q$ consist of a single depot $v(0)$, and the set $R$ of requests in $G(p, \lambda)$ consist of the following $c$ subsets $R[i]$, $i = 1, \ldots, c$, containing $2^{i-1}$ requests. Fig. 5.5 illustrates set $R$ of requests in $G(p, \lambda)$ with $p = 15$. Let set $R[1]$ consist of a single request $r$ with $r^+ = v(0)$ , $r^- = v(2^{c-1})$. Let set $R[2]$ consist of two requests whose travel cost is the half of that of the request in $R[1]$, and expressed by $R[2] = \{r \mid (r^+, r^-) \in \{(v(0), v(2^{c-2})), (v(2^{c-1}), v(2^{c-1}))\}\}$. The $i$-th set $R[i]$, $i = 1, 2, \ldots, c$, consists of $2^{i-1}$ requests and expressed by

$$\begin{aligned} R[i] = \{r \quad | \quad & (r^+, r^-) \in \{(v(0), v(2^{c-i})), (v(2^{c-i}), v(2 \cdot 2^{c-i})), \\ & (v(2 \cdot 2^{c-i}), v(3 \cdot 2^{c-i})), \ldots, (v(2^{c-1} - 2^{c-i}), v(2^{c-1}))\}\}. \end{aligned}$$

Let $q(r) = 1$ for all request $r \in R$.

Given two vertices $v(i), v(j) \in V$ $(i < j)$, let $P(v(i), v(j))$ denote the path from vertex $v(i)$ to vertex $v(j)$, i.e., $P(v(i), v(j)) = [v(i), v(i+1), \ldots, v(j)]$, and $P(v(j), v(i))$ denote the path from vertex $v(j)$ to vertex $v(i)$, i.e., $P(v(j), v(i)) = [v(j), v(j-1), \ldots, v(i)]$.

**Lemma 5.4.** *An optimal PDP route $\sigma$ to instance $G(p, \lambda)$ has travel cost $2\lambda$.*

**Proof:** Any vehicle must start from depot $v(0)$, visit vertex $v(2^{c-1})$, and return to depot $v(0)$, requiring at least $2\lambda$ travel cost. Hence route $\sigma$ consists of path $P(v(0), v(2^{c-1}))$ and path $P(v(2^{c-1}), v(0))$. On the route $\sigma$, a vehicle starts at depot $v(0)$, and serves requests in $R$ along path $P(v(0), v(2^{c-1}))$, and finally returns from vertex $v(2^{c-1})$ to vertex $v(0)$. When serving requests in $R$ on each vertex $v$ along path $P(v(0), v(2^{c-1}))$, a vehicle first delivers requests whose delivery point is vertex $v$, and picks up requests whose pickup point is vertex $v$. The total quantity of requests loaded on a vehicle is less than or equal to capacity $c$ any time on path $P(v(0), v(2^{c-1}))$ since requests on a vehicle contains at most one request from each set $R[i]$, $i = 1, \ldots, c$. The route $\sigma$ is feasible since it satisfies the capacity constraint, each request is served by one vehicle, and the pickup action of each request is taken before its delivery action. □

### 5.5.1 lower bound on loaded travel cost of PDPCMT solutions

This subsection analyzes a lower bound on the loaded travel cost of PDPCMT solutions by using Theorem 5.1. We analyze a lower bound on the loaded travel cost that does not depend on sequences in trips but are derived from travel cost between pickup points and delivery points of requests in the trips. We explain the cost in detail as follows.

We assign a set of integers to each request in $R$. Let $\varphi(r^+, r^-)$ denote a set of integers that are assigned to request $r = (r^+, r^-)$. We first assign a set of integers to requests in $R[c] = \{r \mid (r^+, r^-) \in \{(v(j-1), v(j)) \mid j = 1, 2, \ldots, 2^{c-1}\}\}$ by

$$\varphi(v(j-1), v(j)) = \{j\}, \quad j = 1, 2, \ldots, 2^{c-1}.$$

As shown in Fig. 5.6, the sets of integers that are assigned to the requests in $R - R[c]$ are defined to be

$$\varphi(v(i), v(j)) = \varphi(v(i), v(i+1)) \cup \varphi(v(i+1), v(i+2)) \cup \cdots \cup \varphi(v(j-1), v(j)).$$

Given a trip $t$ in a PDPCMT route, we define cost $cost_b(R(t))$ of a set $R(t)$ of requests that are served along trip $t$ by

$$cost_b(R(t)) = |\bigcup_{r \in R(t)} \varphi(r^+, r^-)|.$$

The following lemma gives a lower bound on the loaded travel cost of a PDPCMT solution.

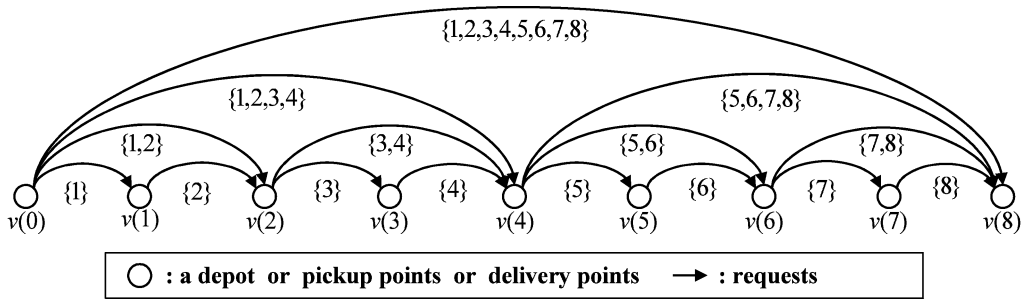



Figure 5.6: A set of integers for each request in $G(p, \lambda)$ with $c = 4$.

**Lemma 5.5.** *Given a PDPCMT solution $s$, the loaded travel cost $cost_f(s)$ of $s$ satisfies*

$$cost_f(s) \geq \sum_{t \in T(s)} cost_b(R(t)) \cdot \lambda / 2^{c-1}.$$

**Proof:** For $i = 1, 2, \ldots, 2^{c-1}$, travel cost $d(v(i-1), v(i))$ along the segment from $v(i-1)$ to $v(i)$ is given by $d(v(i-1), v(i)) = \lambda / 2^{c-1}$. It costs at least $cost_b(R(t)) \cdot \lambda / 2^{c-1}$ to serve the requests in $R(t)$ since a vehicle that serves the requests in $R(t)$ is required to travel at least $cost_b(R(t))$ segments. Thus we have the lemma. □

We derive a lower bound on $cost_f(s)$ over all PDPCMT solution $s$ as follows. For a PDPCMT solution $s$, let $R_1, R_2, \ldots, R_n$ of $R$ be sets of requests such that each $R_i$ corresponds to the set of requests that are served in a trip in solution $s$. Hence it holds $\sum_{r \in R_i} q(r) \leq c$ for $i = 1, 2, \ldots, n$.

We show a lower bound on $\sum_{i=1}^{n} cost_b(R_i)$ by converting a problem that asks to find a partition $s^* = \{R_1, R_2, \ldots, R_n\}$ of $R$ in the binary tree $\mathcal{T}_c$ for PPBT. We obtain the following lemma.

**Lemma 5.6.** *Given a set $R$ of requests in $G(p, \lambda)$, the problem of finding a partition of $R$ can be reduced to PPBT in a polynomial time.*

**Proof:** We associate request $r = (v((j-1) \cdot 2^{c-i}), v(j \cdot 2^{c-i}))$ in $G(p, \lambda)$ with vertex $v((j-1) \cdot 2^{c-i}, j \cdot 2^{c-i})$ on the binary tree for PPBT. Then it holds

$$\varphi(v((j-1) \cdot 2^{c-i}), v(j \cdot 2^{c-i})) = |L(v((j-1) \cdot 2^{c-i}, j \cdot 2^{c-i}))|$$

for $i = 1, 2, \ldots, c$ and $j = 1, 2, \ldots, 2^{i-1}$. If we find a partition $V_1, V_2, \ldots, V_n$ of $\mathcal{T}_c$ to PPBT, then each set $V_i$ of vertices satisfies the capacity constraint. Since each set $V_i$ of vertices in PPBT corresponds to set $R_i$ of requests in $G(p, \lambda)$, each set $R_i$ of requests also satisfies the capacity constraint. Thus we have the lemma. □

We finally prove Theorem 5.4.

**Proof of Theorem 5.4:** A lower bound on $\sum_{t \in T(s)} cost_b(R(t)) = \sum_{i=1}^{n} cost_b(R_i)$ of a

PDPCMT solution $s$ is obtained by solving PPBT that is obtained by the reduction in Lemma 5.6. Let $s^* = \{R_1, R_2, \ldots, R_n\}$. By using Theorem 5.1, we have

$$\sum_{t \in T(s)} cost_b(R(t)) = cost_b(s^*) \geq 2^{c-2}c/h.$$

Thus we have

$$\begin{aligned} opt_{\text{PDPCMT}}(G(p,\lambda)) &\geq cost_f(s) \geq \sum_{t \in T(s)} cost_b(R(t)) \cdot \lambda/2^{c-1} \\ &\geq 2^{c-2}c/h \cdot \lambda/2^{c-1} \\ &= \lambda c/(2\log_2 c) \\ &= \lambda \log_2(p+1)/(2\log_2\log_2(p+1)). \end{aligned}$$

By using $opt_{\text{PDP}}(G(p,\lambda)) = 2\lambda$, we have the theorem. □

## 5.6 Analyses on the optimal cost between PDPTCMT and PDPT

We furthermore show the following theorem.

**Theorem 5.5.** *Let $I = (Q, R, c, t)$ be an instance with $p \geq 2$ requests. Suppose that there is a transshipment point $t$, and each vehicle can visit $t$ any number of times. In PDPTCMT, each vehicle is allowed to visit $t$ after the vehicle visits all pickup points of its requests and before it visits the corresponding delivery points during its trip. Then it holds*

$$opt_{\text{PDPTCMT}}(I) \leq \lceil \log_2 4p \rceil \cdot opt_{\text{PDPT}}(I).$$

□

For this, we show how to convert a PDPT solution to a PDPTCMT solution. Since there exists only one transshipment point $t$, each request is dropped at $t$ at most once. In this subsection, if a request $r$ is served by temporarily dropped and picked up at transshipment point $t$ in a PDPT solution $s$, then we regard that the request $r$ consists of two requests $r_1$ and $r_2$ such that $r_1^+ = r^+, r_2^- = r^-$, and $r_1^-$ and $r_2^+$ are taken at transshipment point $t$. We call the original request $r$ an *input request* and call the two request $r_1$ and $r_2$ *split requests*. Thus an input request that is served by temporarily dropped and picked up at $t$ makes four actions in solution $s$. Since split requests $r_1$ and $r_2$ are served by one vehicle respectively in solution $s$, we can regard PDPT solution $s$ as a PDP solution that satisfies an additional requirement that $r_1^-$ is required to be taken before $r_2^+$. ($r_1$ and $r_2$ may be served by different vehicles.)

**Theorem 5.6.** *Given a PDPT route $\sigma$ to an instance $I = (Q, R, c, t)$ with $p'$ input requests, PDPTCMT routes $\pi_1, \pi_2, \ldots, \pi_k$ with*

$$\bigcup_{i=1}^{k} A(\pi_i) = A(\sigma) \text{ and } \sum_{i=1}^{k} cost(\pi_i) \leq \lceil \log_2 |A(\sigma)| \rceil \cdot cost(\sigma)$$

*can be constructed in polynomial time, where it holds $|A(\sigma)|/4 \leq p'$.*

**Proof:** As described above, route $\sigma$ is regarded as a route in a PDP solution that satisfies a requirement that $r_1^-$ is required to be taken before $r_2^+$ for all pairs of split requests $r_1$ and $r_2$. We apply the algorithm that is used to prove Theorem 4.2 to $\sigma$, and obtain routes $\pi_1, \pi_2, \ldots, \pi_k$, which satisfy

$$\bigcup_{i=1}^{k} A(\pi_i) = A(\sigma) \text{ and } \sum_{i=1}^{k} cost(\pi_i) \leq \lceil \log_2 |A(\sigma)| \rceil \cdot cost(\sigma).$$

By Lemma 5.3, routes $\pi_1, \pi_2, \ldots, \pi_k$ are PDPCMT routes. It suffices to show that $r_1^-$ is taken before $r_2^+$ on $\pi_1, \pi_2, \ldots, \pi_k$ for each input request $r$ that is served by temporarily dropped and picked up at $t$ on $\sigma$. Since the order of actions that are taken on route $\pi_i$ is the same with the order on $\sigma$ for $i = 1, \ldots, k$, and $r_1^-$ is taken before $r_2^+$ on $\sigma$, routes $\pi_i$, $i = 1, \ldots, k$, can be travelled by $k$ vehicles, so that $r_2^+$ is taken after $r_1^-$ is finished. Thus we have the theorem. □

A lower bound on the optimal cost $opt_{\text{PDPTCMT}}(I)$ of PDPTCMT solutions can be derived from the lower bound on the optimal cost $opt_{\text{PDPCMT}}(I)$ of PDPCMT solutions. The lower bound on $opt_{\text{PDPCMT}}(I)$ holds between PDPTCMT and PDPT by locating a transshipment point at a place that is enough far from a depot so that the transshipment point does not contribute to decrease of travel cost. Analogously with Theorem 5.4, we obtain the following theorem.

**Theorem 5.7.** *For any integer $h \geq 1$, there exists an instance $I = (Q, R, c)$ with $p = 2^{2^h} - 1$ requests that satisfies*

$$opt_{\text{PDPTCMT}}(I) \geq \frac{\log_2(p+1)}{4 \log_2 \log_2(p+1)} \cdot opt_{\text{PDPT}}(I).$$

□

# Chapter 6

# Conclusion

Throughout this thesis, we have made two types of contributions for decreasing physical distribution cost.

One contribution is proposing a fast construction algorithm that finds a cheaper routing schedule than a current one. We considered the discrete split delivery vehicle routing problem (DSDVRP), and proposed a fast algorithm that constructs routes one by one without any improvement procedures to the DSDVRP. The algorithm generates routes by a dynamic programming based on an elaborate route evaluation function that estimates the total travel cost that is required to serve all the remaining items by vehicles.

In Chapter 2, we made preparations to show the algorithm for DSDVRP. We first defined problems VRP and SDVRP, and described standard construction algorithms for VRP; the saving algorithm, the insertion algorithm, and the sweep algorithm. For SDVRP, we introduced Dror and Trudeau's heuristic algorithm. Those algorithms were used to compare solution quality with that of our algorithm for DSDVRP in Chapter 3.

In Chapter 3, we explained the detail of the construction algorithm for DSDVRP. We then conducted computational experiments, and showed that our algorithm is practically efficient and fast comparing to the representative heuristics for VRP and SDVRP. Our approach to estimate the total travel cost of resulting items accurately would be applied to other types of routing problems in order to produce good quality solutions with short computation time. Furthermore mixing several improvement methods including sophisticated metaheuristics with our construction algorithm could produce good solutions with less travel cost.

The other contribution is analyzing the maximum possible cost that can be saved by relaxing some constraints. As described in Section 1.1, it is desirable to know the possible cost saving by each type of constraints on routing without conducting accurate estimations of their performances.

In Fig. 1.4 in Chapter 1, we showed factors of upper and lower bounds on the maximum

cost that can be saved by regarding an instance to problem A as that to problem B, where A (resp., B) corresponds to the problem that is located on the head (resp., tail) of each arc.

In this thesis, we made the following analyses. In Chapter 4, we studied the maximum cost that can be saved by introducing a transshipment point to PDP. We showed that

$$opt_{\mathrm{PDP}}(I) \leq (6\lceil p^{1/2} \rceil + 1) \cdot opt_{\mathrm{PDPT}}(I)$$

holds for any instance $I$ with $p$ requests and one transshipment point if each vehicle can visit the transshipment point at most once. The inequality is shown by presenting a polynomial time algorithm that converts a solution to PDPT to a solution to PDP, and by showing that the travel cost of the constructed solution is bounded by the ratio. We then showed that if each vehicle can visit the transshipment point any number of times, then it holds

$$opt_{\mathrm{PDP}}(I) \leq (6\lceil p^{1/2} \rceil + 2) \cdot opt_{\mathrm{PDPT}}(I).$$

Furthermore that the bound is valid for the ratio of $opt_{\mathrm{PDPC}}(I)$ to $opt_{\mathrm{PDPTC}}(I)$, and we present an instance $I'$ that satisfies $opt_{\mathrm{PDPC}}(I') = p^{1/4} \cdot opt_{\mathrm{PDPTC}}(I')$.

In Chapter 5, we examined how the least travel cost can be increased by the requirement to PDP that no vehicle which has begun a delivery action is allowed to take pickup actions until all of the loads on the vehicle are delivered. We showed that the maximum ratio of the optimal value of PDPCMT to that of PDP over all instances with $p$ requests is presented by

$$opt_{\mathrm{PDPCMT}}(I) \leq \lceil \log_2 2p \rceil \cdot opt_{\mathrm{PDP}}(I).$$

The analysis for the upper bound could be easily extended to a case with a transshipment point. We showed that the maximum ratio of the optimal value of PDPTCMT to that of PDPT over all instances with $p$ requests is given by

$$opt_{\mathrm{PDPTCMT}}(I) \leq \lceil \log_2 4p \rceil \cdot opt_{\mathrm{PDPT}}(I).$$

We furthermore presented an instance $I$ that gives a lower bound that satisfies

$$opt_{\mathrm{PDPCMT}}(I) \geq \frac{\log_2(p+1)}{4 \log_2 \log_2(p+1)} \cdot opt_{\mathrm{PDP}}(I).$$

The lower bound holds for the case with a transshipment point by regarding that the transshipment point is located at a sufficiently far location such that using the transshipment point does not contribute to reduce the total cost. Thus there is an instance $I$ that satisfies

$$opt_{\mathrm{PDPTCMT}}(I) \geq \frac{\log_2(p+1)}{4 \log_2 \log_2(p+1)} \cdot opt_{\mathrm{PDPT}}(I).$$

In order to show the lower bound, we first introduced a partition problem that asks to find a partition of vertices on a complete binary tree with the minimum cost, and presented a lower

bound on the partition problem. We analyzed a lower bound on travel cost of a PDPCMT solution by considering travel cost that does not depend on sequences on trips but are derived from travel cost between pickup points and delivery points of requests in the trips. A lower bound of the travel cost was obtained by using the result of the lower bound for the partition problem.

We mention some subjects for future works. One subject is that developing efficient algorithms for the pickup and delivery problem for transfer (PDPT) and the multi-trip pickup and delivery problem for consecutive pickups and deliveries (PDPCMT). We encounter these problems much often in practical cases, however few papers that propose algorithms for these problems are reported. Furthermore the effectiveness on reducing travel cost by transferring requests at a transshipment point is high comparing to admitting split delivery to the VRP. Developing efficient algorithms for constructing PDPT routes would be practically helpful to reduce distribution costs in logistics and transportation industries. Another subject is that developing algorithms and analyzing optimal costs for problems that consider multiple constraints such as split deliveries, transfers, time windows, and consecutive pickups and deliveries. Most routing problems in practical cases include such multiple constraints, and it is sometimes difficult to obtain even feasible solutions.

Due to the enhancement of information technology, applications of distribution systems that incorporate efficient algorithms for VRP and PDP have been developed in several industries, and they have been making drastic improvement on decreasing distribution cost and CO2 emission. Furthermore excellent theoretical analyses would contribute to making an important decision on changes to the current routing rules and the existing facilities. The author hopes that the work contained in this thesis will be helpful to the study and the application to a real world in this field.

# Bibliography


[1] E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, John Wiley and Sons, 1997.

[2] U. Aickelin and K. A. Dowsland, An indirect genetic algorithms for a nurse-scheduling problem, *Computers & Operations Research*, Vol. 31, No. 5, pp. 761-778, 2004.

[3] R. E. Aleman, X. Zhang, and R. R. Hill, An adaptive memory algorithm for the split delivery vehicle routing problem, *Journal of Heuristics*, Published Online, 11 December 2008.

[4] C. Archetti, M. G. Speranza, and A. Hertz, A tabu search algorithm for the split delivery vehicle routing problem, *Transportation Science*, Vol. 40, No. 1, pp. 64–73, 2006.

[5] C. Archetti, M. W. P. Savelsbergh, and M. G. Speranza, Worst-case analysis for split delivery vehicle routing problems, *Transportation Science*, Vol. 40, No. 2, pp. 226–234, 2006.

[6] E. Arkin, R. Hassin, and L. Klein, Restricted delivery problems on a network, *Networks*, Vol. 29, pp. 205–216, 1997.

[7] F. Bellanti, G. Carello, F. D. Croce, and R. Tadei, A greedy-based neighborhood search approach to a nurse rostering problem, *European Journal of Operational Research*, Vol. 153, No. 1, pp. 28–40, 2004.

[8] J. M. Belenguer, M. C. Martinez, and E. Mota, A lower bound for the split delivery vehicle routing problem, *Operations Research*, Vol. 48, pp. 801–810, 2000.

[9] R. Bent and P. Van Hentenryck, A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows, *Proceedings of the International Conference on Constraint Programming (CP-2003)*, pp. 123–137, 2003.

[10] L. Bodin, B. Golden, A. Assad, and M. Ball, The state of the art in the routing and scheduling of vehicles and crews, *Computers and Operations Research*, special issue, June 1983.

[11] R. L. Bowerman, P. H. Calamai, and G. Brent Hall, The spacefilling curve with optimal partitioning heuristic for the vehicle routing problem, *European Journal of Operational Research*, Vol. 76, pp. 128–142, 1994.

[12] O. Bräysy, A reactive variable neighborhood search for the vehicle routing problem with time windows, *INFORMS Journal on Computing*, Vol. 15, No. 4, pp. 347–368, 2003.

[13] O. Bräysy and M. Gendreau, Vehicle routing problem with time windows, part I: Route construction and local search algorithms, *Transportation Science*, Vol. 39, No. 1, pp. 104–118, 2005.

[14] L. I. Burke, Neural methods for the traveling salesman problem: Insights from operations research, *Neural Networks*, Vol. 7, No. 4, pp. 681–690, 1994.

[15] A. M. Campbell and M. Savelsbergh, Efficient insertion heuristics for vehicle routing and scheduling problems, *Transportation Science*, Vol. 38, No. 3, pp. 369–378, 2004.

[16] M. Charikar, S. Khuller, and B. Raghavachari, Algorithms for capacitated vehicle routing, *STOC*, pp. 349–358, 1998.

[17] S. Chen, B. Golden, and E. Wasil, The split delivery vehicle routing problem: Applications, algorithms, test problems, and computational results, *Networks*, Vol. 49, No. 4, pp. 318–329, 2007.

[18] G. Clarke and J. Wright, Scheduling of vehicles from a central depot to a number of delivery points, *Operations Research*, Vol. 12, No. 4, pp. 568–582, 1964.

[19] C. E. Cortés, M. Matamala, and C. Contardo, The pickup-and-delivery problem with transfers: Formulation and a branch-and-cut solution method, *European Journal of Operational Research*, Vol. 200, No. 3, pp. 711–724, 2010.

[20] H. K. Chung and J. P. Norback, A clustering and insertion heuristic applied to a large routing problem in food distribution, *The Journal of the Operational Research Society*, Vol. 42, No. 7, pp. 555–564, 1991.

[21] G. A. Croes, A method for solving traveling-salesman problems, *Operations Research*, Vol. 6, No. 6, pp. 791–812, 1958.

[22] G. B. Dantzig and J. H. Ramser, The truck dispatching problem, *Management Science*, Vol. 6, No. 1, pp. 80–91, 1959.

[23] M. Dell'Amico, M. Fischetti, and P. Toth, Heuristic algorithms for the multiple depot vehicle scheduling problem, *Management Science*, Vol. 39, No. 1, pp. 115–125, 1993.

[24] D. Desaulniers, J. Desrosiers, A. Erdmann, M. M. Solomon, and F. Soumis, VRP with pickup and delivery, In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 9, pp. 225–242, Society for Industrial and Applied Mathematics, 2002.

[25] M. Desrochers, J. K. Lenstra, M. W. P. Savelsbergh, and F. Soumis, Vehicle routing with time windows: Optimization and approximation, In B. L. Golden and A. A. Assad, editors, *Vehicle Routing: Methods and Studies*, Studies in management science and systems, pp. 65–84, Noth-Holland, Amsterdam, 1988.

[26] M. Desrochers, Y. Dumas, M. M. Solomon, and F. Soumis, Time constrained routing and scheduling, In M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 2, pp. 35–139, Noth-Holland, Amsterdam, 1995.

[27] K. A. Dowsland and J. M. Thompson, Solving a nurse scheduling problem with knapsacks, networks and tabu search, *Journal of the Operational Research Society*, Vol. 51, pp. 825–833, 2000.

[28] M. Dror and P. Trudeau, Savings by split delivery routing, *Transportation Science*, Vol. 23, No. 2, pp. 141–145, 1989.

[29] M. Dror and P. Trudeau, Split delivery routing, *Naval Research Logistics*, Vol. 37, pp. 383–402, 1990.

[30] M. Dror, G. Laporte, and P. Trudeau, Vehicle routing with split deliveries, *Discrete Applied Mathematics*, Vol. 50, pp. 239–254, 1994.

[31] P. G. Eibl, R. Mackenzie, and D. B. Kidner, Vehicle routing and scheduling in the brewing industry: A case study, *International Journal of Physical Distribution and Logistics Management*, Vol. 24, No. 6, pp. 27–37, 1994.

[32] M. L. Fisher and R. Jaikumar, A generalized assignment heuristic for vehicle routing, *Networks*, Vol. 11, No. 2, pp. 109–124, 1981.

[33] A. M. Frisch, T. J. Peugniez, A. J. Doggett, and P. W. Nightingale, Solving non-boolean satisfiability problems with stochastic local search: A comparison of encodings, *Journal of Automated Reasoning*, Vol. 35, No. 1-3, pp. 143–179, 2005.

[34] P. W. Frizzell and J. W. Giffin, The bounded split delivery vehicle routing problem with grid network distances, *Asia-Pacific Journal of Operational research*, Vol. 9, pp. 101–116, 1992.

[35] P. W. Frizzell and J. W. Giffin, The split delivery vehicle scheduling problem with time windows and grid network distances, *Computers and Operations Research*, Vol. 22, pp. 655–667, 1995.

[36] W. W. Garvin, H. W. Crandall, J. B. John, and R. A. Spellman, Applications of linear programming in the oil industry, *Management Science*, Vol. 3, No. 4, pp. 407–430, 1957.

[37] M. Gendreau, A. Hertz, and G. Laporte, A tabu search heuristic for the vehicle routing problem, *Management Science*, Vol. 40, No. 10, pp. 1276–1290, 1994.

[38] M. Gendreau, G. Laporte, and J. Y. Potvin, Vehicle routing: Modern heuristics, E. H. Aarts, J. K. Lenstra, eds. *Local Search in Combinatorial Optimization*, Wiley, Chichester, UK, pp. 311-336, 1997.

[39] B. Gillet and L. Miller, A heuristic algorithm for the vehicle dispatch problem, *Operations Research*, Vol. 22, No. 2, pp. 340–349, 1974.

[40] B. L. Golden, E. A. Wasil, J. P. Kelly, and I. Chao, The impact of metaheuristics on solving the vehicle routing problem: Algorithms, problem sets, and computational results, T. G. Cranic, G. Laporte, eds. *Fleet Management and Logistics*, Boston, MA, pp. 33-56, 1998.

[41] J. Hansen and B. Jaumard, Algorithms for the maximum satisfiability problem, *Computing*, Vol. 44, pp. 279–303, 1990.

[42] H. Hashimoto, T. Ibaraki, S. Imahori, and M. Yagiura, The vehicle routing problem with flexible time windows and traveling times, *Discrete Applied Mathematics*, Vol. 154, No. 16, pp. 2271–2290, 2006.

[43] H. H. Hoos and T. Stutzle, *Stochastic Local Search: Foundations and Applications*, Morgan Kaufmann, 2005.

[44] Japan institute of logistics systems, Investigation on Physical Distribution Cost (in Japanese), 2008.

[45] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, Worst-case performance bounds for simple one-dimensional packing algorithms, *SIAM Journal on Computing*, Vol. 3, No. 4, pp. 299–325, 1974.

[46] D. S. Johnson and L. A. McGeoch, The traveling salesman problem: A case study in local optimization, In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, John Wiley and Sons, 1997.

[47] G. A. P. Kindervater and M. W. P. Savelsbergh, Vehicle routing: handling edge exchanges, In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pp. 337–360, John Wiley and Sons, 1997.

[48] G. Laporte, M. Gendreau, J. Y. Potvin, and F. Semet, Classical and modern heuristics for the vehicle routing problem, *International Transactions in Operational Research*, Vol. 7, No. 4/5, pp. 285–300, 2000.

[49] G. Laporte, Vehicle Routing, In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, chapter 14, pp. 223–240, John Wiley and Sons, 1997.

[50] G. Laporte and I. H. Osman, Routing problems: A bibliography, *Annals of Operations Research*, Vol. 61, pp. 227–262, 1995.

[51] H. Li and A. Lim, A metaheuristic for the pickup and delivery problem with time windows, *International Journal on Artificial Intelligence Tools*, Vol. 12, No. 2, pp. 173–186, 2003.

[52] S. Lin, Computer solutions of the traveling salesman problem, *Bell System Technical Journal*, Vol. 44, pp. 2245–2269, 1965.

[53] S. Lin and B. Kernighan, An effective heuristic algorithm for the traveling-salesman problem, *Operations Research*, Vol. 21, No. 2, pp. 498–516, 1973.

[54] K. Liu, A study on the split delivery vehicle routing problem, *Ph.D. Dissertation, Department of Industrial Engineering, Mississippi State University, Mississippi State, Mississippi*, 2005.

[55] Ministry of the Environment, Government of Japan, Annual report on the environment, 2008.

[56] S. Mitrović-Minić and G. Laporte, The pickup and delivery problem with time windows and transshipment, *INFOR*, Vol. 44, No. 3, pp. 217–227, 2006.

[57] N. Mladenović and P. Hansen, Variable neighborhood search, *Computers & Operations Research*, Vol. 24, No. 11, pp. 1097–1100, 1997.

[58] P. A. Mullaseril, M. Dror, and J. Leung, Split-delivery routing heuristics in livestock feed distribution, *Journal of the Operational Research Society*, Vol. 48, pp. 107–116, 1997.

[59] G. Nagy and S. Salhi, Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries, *European Journal of Operational Research*, Vol. 162, pp. 126–141, 2005.

[60] Y. Nakao and H. Nagamochi, A DP-based heuristic algorithm for the discrete split delivery vehicle routing problem, IN *Proceedings of the International Symposium on Scheduling 2006 (ISS2006)*, pp. 42–47, 2006.

[61] Y. Nakao and H. Nagamochi, Worst case analysis for pickup and delivery problems with consecutive pickups and deliveries, *The 20th International Symposium on Algorithms and Computation (ISAAC), Lecture Notes in Computer Science*, pp. 554–563, 2009.

[62] Y. Nakao and H. Nagamochi, Worst case analysis for pickup and delivery problems with transfer, *IEICE Transaction*, Vol. E91-A, No. 9, pp. 2328–2334, 2008.

[63] W. P. Nanry and J. W. Barnes, Solving the pickup and delivery problem with time windows using reactive tabu search, *Transportation Research Part B*, Vol. 34, No. 2, pp. 107–121, 2000.

[64] M. A. Nowak, The pickup and delivery problem with split loads, *Ph.D. Thesis, Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia*, 2005.

[65] J. Oppen and A. Lokketangen, Transportation of livestock to slaughterhouse, In *Proceedings of the Sixth Metaheuristics International Conference (MIC2005)*, pp. 719–724, 2005.

[66] I. Or, Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking, *Ph.D. Thesis, Department of Industrial Engineering and Management Science, Northwestern University, Evanston, IL*, 1976.

[67] H. Paessens, The savings algorithm for the vehicle routing problem, *European Journal of Operational Research*, Vol. 34, pp. 336–344, 1988.

[68] G. Pankratz, A grouping genetic algorithm for the pickup and delivery problem with time windows, *OR Spectrum*, Vol. 27, No. 1, pp. 21–41, 2005.

[69] C. H. Papadimitriou and K. Steiglitz, Combinatorial Optimization, *Algorithms and Complexity*, Dover, 1998.

[70] R. J. Petch and S. Salhi, A multi-phase constructive heuristic for the vehicle routing problem with multiple trips, *Discrete Applied Mathematics*, Vol. 133, No. 1, pp. 69–92, 2004.

[71] J. Privé, J. Renaud, F. Boctor, and G. Laporte, Solving a vehicle-routing problem arising in soft-drink distribution, *Journal of the Operational Research Society*, Vol. 57, pp. 1045–1052, 2006.

[72] G. Reinelt, TSPLIB - A traveling salesman problem library, *ORSA Journal on Computing*, Vol. 3, pp. 376–384, 1991.

[73] S. Richter, M. Helmert, and C. Gretton, A stochastic local search approach to vertex cover, *Proceedings of the 30th annual German conference on Advances in Artificial Intelligence*, September 10-13, pp. 412–426, 2007.

[74] S. Ropke and D. Pisinger, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows, *Technical Report*, Department of Computer Science, University of Copenhagen, 2004.

[75] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis II, An analysis of several heuristics for the traveling salesman problem, *SIAM Journal on Computing*, Vol. 6, No. 3, pp. 563–581, 1977.

[76] M. W. P. Savelsbergh and M. Sol, The general pickup and delivery problem, *Transportation Science*, Vol. 29, No. 1, pp. 17–29, 1995.

[77] M. W. P. Savelsbergh and M. Sol, DRIVE: Dynamic routing of independent vehicles, *Operations Research*, Vol. 46, No. 4, pp. 474–490, 1998.

[78] J. S. Shang and C. K. Cuff, Multicriteria pickup and delivery problem with transfer opportunity, *Computers and Industrial Engineering*, Vol. 30, No. 4, pp. 631–645, 1996.

[79] S. J. Shyu, P. Y. Yin, and B. M. T. Lin, An ant colony optimization algorithm for the minimum weight vertex cover problem, *Annals of Operations Research*, Vol. 131, No. 1-4, pp. 283–304, 2004.

[80] G. Sierksma and G. A. Tijssen, Routing helicopters for crew exchanges on off-shore locations, *Annals of Operations Research*, Vol. 76, pp. 261–286, 1998.

[81] M. Sigurd, D. Pisinger, and M. Sig, Scheduling transportation of live animals to avoid the spread of diseases, *Transportation Science*, Vol. 38, No. 2, pp. 197–209, 2004.

[82] M. M. Solomon, Algorithms for the vehicle routing and scheduling problem with time window constraints, *Operations Research*, Vol. 35, No. 2, pp. 254–265, 1987.

[83] M. M. Solomon and J. Desrosiers, Time window constrained routing and scheduling problems, *Transportation Science*, Vol. 22, No. 1, pp. 1–13, 1988.

[84] S. H. Song, K. S. Lee, and G. S. Kim, A practical approach to solving a newspaper logistics problem using a digital map, *Computers and Industrial Engineering*, Vol. 43, No. 1-2, pp. 315–330, 2002.

[85] E. Taillard, Parallel iterative search methods for vehicle routing problems, *Networks*, Vol. 23, No. 8, pp. 661–673, 1993.

[86] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J. Y. Potvin, A tabu search heuristic for the vehicle routing problem with soft time windows, *Transportation Science*, Vol. 31, No. 2, pp. 170–186, 1997.

[87] P. Toth and D. Vigo, An overview of vehicle routing problems, In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 1, pp. 1–26, Society for Industrial and Applied Mathematics, 2002.

[88] P. Toth and D. Vigo. editors, The vehicle routing problem, *Society for Industrial and Applied Mathematics*, 2002.

[89] P. Toth and D. Vigo, The granular tabu search and its application to the vehicle-routing problem, *INFORMS Journal on Computing*, Vol. 15, No. 4, pp. 333–346, 2003.

[90] P. Toth and D. Vigo, VRP with backhauls, In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 8, pp. 195–224, Society for Industrial and Applied Mathematics, 2002.

[91] A. van Vliet, C. G. E. Boender, and A. H. G. Rinnooy Kan, Interactive optimization of bulk sugar deliveries, *Interfaces*, Vol. 22, No. 3, pp. 4–14, 1992.

[92] M. Yagiura and T. Ibaraki, Local search, In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of Applied Optimization*, pp. 104–123, Oxford University Press, New York, 2002.

[93] M. Yannakakis, The analysis of local search problems and their heuristics, *Proceedings of the Seventh Annual Symposium on Theoretical Aspects of Computer Science*, pp. 298–311, 1990.